UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR

POSGRADOS

MAESTRÍA EN

SOFTWARE CON MENCIÓN EN DISEÑO DE ARQUITECTURA DE SISTEMAS

RPC-SO-34-NO.778-2021

OPCIÓN DE TITULACIÓN:

PROYECTO DE TITULACIÓN CON COMPONENTES DE INVESTIGACIÓN APLICADA Y/O DE DESARROLLO

TEMA:

INTEGRACIÓN DE GRAPHQL Y SEGURIDAD ZERO TRUST EN EL DISEÑO DE APIS

AUTOR

GUIDO GONZALO GUERRERO DURÁN

DIRECTOR:

JOE FRAND LLERENA IZQUIERDO

QUITO – ECUADOR

2025

Autor:

Guido Gonzalo Guerrero Durán
Ingeniero en Sistemas
Candidato a Magíster en Software por la Universidad Politécnica Salesiana – Sede Quito.
gguerrerod@est.ups.edu.ec

Dirigido por:

Joe Frand Llerena Izquierdo
Ingeniero en Computación
Magister en Sistemas de Información Gerencial
jllerena@ups.edu.ec

DEDICATORIA

Dedico este trabajo principalmente a Dios, por haberme dado la vida y permitirme el haber llegado hasta este momento tan importante de mi formación profesional.

A Myriam, mi esposa, por ser el pilar más importante y por demostrarme siempre su cariño y apoyo incondicional sin importar nuestras diferencias de opiniones.

A Gonzalo, cuya alegría y tenacidad me hacen recordar el verdadero significado de la vida.

A mi pequeño Pablito Andrés, partiste demasiado pronto, pero sé que siempre me observas y tu luz guiará mi camino el resto de mi vida.

Este proyecto es el resultado de cada sacrificio, de cada noche en vela y de cada sueño compartido. A todos ustedes, con amor y gratitud eterna.

Guido Guerrero D.

AGRADECIMIENTO

Agradezco a mi esposa e hijos, que siempre me han brindado su apoyo para poder cumplir todos mis objetivos personales y académicos. Ellos con su cariño me han impulsado siempre a perseguir mis metas y nunca abandonarlas frente a las adversidades.

Agradezco también a mi director de tesis, Ingeniero Joe Frand Llerena Izquierdo, por su valiosa guía, conocimientos y apoyo a lo largo de esta investigación. Su orientación ha sido importante para el desarrollo de este trabajo.

Finalmente, agradezco a mis compañeros de la maestría por sus ideas y sugerencias, así como a los autores y fuentes que han contribuido al marco teórico de este estudio.

Tabla de Contenido

Tabla de Ilustraciones

INTEGRACIÓN DE GRAPHQL Y SEGURIDAD ZERO TRUST EN EL DISEÑO DE APIS

AUTOR:

GUIDO GONZALO GUERRERO DURÁN

Resumen

Este trabajo de titulación se centra en el diseño de APIs GraphQL y como implementar el modelo Zero Trust en la misma. Por tanto, el objetivo principal es proponer la integración del modelo Zero Trust en el API GraphQL de tal manera que el rendimiento y velocidad de respuesta del API no se vea afectada, a la vez, que la seguridad ha mejorado, respetando el principio que indica que las amenazas pueden estar dentro y/o fuera de la red, aplicando una verificación rigurosa durante cada acceso o consulta a GraphQL.

Las principales ventajas de GraphQL son su flexibilidad y eficiencia al momento de obtener información, sin embargo, estas ventajas también exponen a las APIs a posibles ataques y es aquí, donde a través de la aplicación del modelo Zero Trust se disminuyen los riesgos mediante la autenticación y autorización en cada acceso, mismos que serán controlados y verificados.

El presente trabajo analiza las amenazas detectadas previamente en las APIs GraphQL, tales como: ataques DOS (Denial of Service), exposición excesiva de datos (overfetching), inyección de GraphQL (GraphQL injection), entre otros. De igual manera, se analiza la seguridad que brinda el modelo Zero Trust y como las ventajas del mismo pueden fortalecer la seguridad de APIs GraphQL. Durante el avance del presente trabajo se desarrollará una prueba de concepto (POC) de una API GraphQL utilizando los principios del modelo Zero Trust para realizar las pruebas y evaluaciones respectivas de seguridad y rendimiento.

Esta investigación combinará las bondades de las APIs GraphQL con las fortalezas del modelo Zero Trust para obtener una API robusta en cuanto a seguridades sirviendo así de base referencial para el diseño y desarrollo de futuros proyectos, donde desarrolladores y administradores puedan beneficiarse de sus fortalezas y optimizaciones de seguridad y rendimiento.

Palabras clave:

GraphQL, Seguridad Zero Trust, Diseño de APIs, Autenticación y autorización, Ciberseguridad.

ABSTRACT

This research focuses on the design of GraphQL APIs and how to implement the Zero Trust model within them. Therefore, the main objective is to propose the integration of the Zero Trust model into the GraphQL API in such a way that the API's performance and response speed are not affected, while also improving security, respecting the principle that indicates that threats can be inside and/or outside the network, applying rigorous verification during each access or query to GraphQL.

The main advantages of GraphQL are its flexibility and efficiency when obtaining information, however, these advantages also expose APIs to possible attacks and it is here, where through the application of the Zero Trust model, risks are reduced through authentication and authorization in each access, which will be controlled and verified.

This work examines previously identified GraphQL APIs' threats, such as Denial of Service (DoS) attacks, excessive data exposure (overfetching), GraphQL injection, and others. Additionally, it analyzes the security enhancements provided by the Zero Trust model and how its advantages can make GraphQL API security stronger. As part of this research, a proof of concept (PoC) will be developed using a GraphQL API based on Zero Trust model and its principles. This PoC will be used to conduct security and performance testing and evaluations.

This research will combine the benefits of GraphQL APIs with the strengths of the Zero Trust model to obtain a robust API in terms of security, thus serving as a reference base for the design and development of future projects, where developers and administrators can benefit from its strengths and security and performance optimizations.

Palabras clave:
GraphQL, Zero Trust, API Design, Authentication, Authorization, Cybersecurity.

1. INTRODUCCIÓN

En la actualidad, las amenazas cibernéticas y las filtraciones de datos están en constante aumento, lo que ha hecho que la protección de las interfaces de programación de aplicaciones (APIs) sea una prioridad en el ámbito de la seguridad de la información.

En 2015, Facebook expuso GraphQL como un nuevo modelo de APIs, para gestionar la consulta y consumo de información. Estas APIs GraphQL permiten consultar únicamente los datos requeridos evitando la consulta de información innecesaria que genera mayor consumo de ancho de banda, especialmente costoso en conexiones limitadas y/o móviles (Pappula & Anasuri, 2021; Shah et al., 2024).

Esta nueva flexibilidad, aún con sus beneficios, presenta grandes desafíos y riesgos en el campo de la seguridad de la información, debido a que un atacante puede obtener más información de la necesaria, provocando una denegación de servicios (DoS) (Belhadi et al., 2024). Con esto en mente, el modelo de seguridad Zero Trust ha nacido como una solución clave para proteger aplicaciones y sistemas modernos. Zero Trust elimina la confianza implícita en cualquier usuario o dispositivo, requiriendo continuamente la autenticación y autorización de sus usuarios, lo que aumenta la resistencia frente a accesos no autorizados y filtraciones de datos (Dhiman et al., 2024; Liu et al., 2021).

Implementar el modelo Zero Trust en una API GraphQL contribuirá a disminuir la superficie de riesgos de ataques (Ghasemshirazi et al., 2023; Denzel, 2025), puesto que, se podrán establecer controles precisos respecto a la seguridad basada en la identidad y el uso que cada usuario da al aplicativo. Así mismo, se espera que el rendimiento del API GraphQL no se vea afectado.

Todos aquellos sistemas informáticos, cuyas arquitecturas sean distribuidas, como los microservicios, se verán beneficiados, puesto que estas arquitecturas al tener

una mayor superficie de riesgo requieren mayores y mejores estrategias de protección en cuanto a ciberseguridad se refiere (Abgaz et al., 2023; Söylemez et al., 2022).

En la actualidad, existen estudios respecto a la implementación de seguridad en sistemas empresariales de alto impacto (Aslan et al., 2023), sin embargo, la integración del modelo Zero Trust en APIs GraphQL es un área por investigar, razón del origen del presente estudio, mismo que servirá de marco teórico y técnico para las organizaciones y empresas que deseen implementar seguridades al diseñar sus APIs GraphQL con el modelo Zero Trust.

2. Determinación del Problema

GraphQL apareció como una alternativa a las APIs ReST tradicionales, donde toda la información era enviada al cliente en un solo bloque, puesto que ahora es posible consultar solo la información requerida optimizando el uso de datos, sin embargo, debido a la posibilidad del usuario de recabar tanta información como exponga el API GraphQL un atacante puede consultar al servicio y este intentará devolver una gran cantidad de información, colapsando el servicio, también conocido como ataque de denegación de servicios (Denial of Service) (Belhadi et al., 2024).

Para disminuir el riesgo, el modelo Zero Trust establece múltiples principios, agnósticos a la tecnología usada, para implementar y robustecer la arquitectura de la aplicación y proteger el acceso a los recursos (Dhiman et al., 2024; Liu et al., 2021). Es requerido entender que para el modelo Zero Trust, todos los orígenes de datos y servicios informáticos son considerados recursos, esto quiere decir que, en arquitecturas distribuidas, cada microservicio es un recurso y es necesario que se autentique y autorice en cada intento de conexión sin importar el origen.

El modelo Zero Trust tiene variaciones de su arquitectura, ya que no es posible aplicar todos sus principios en una sola implementación, sino que deben implementarse según la necesidad de cada aplicación.

Se debe mencionar que, aunque han existido estudios por separado, la integración del modelo Zero Trust en las APIs GraphQL ha sido poco explorado y menos aún documentado generando un vacío de conocimiento y técnico en las prácticas empresariales, puesto que, casi todas las APIs GraphQL existentes no cuentan con seguridades eficientes independientemente del aplicativo o usuario que intente acceder al API.

Esta ausencia de información teórica y práctica que describa como implementar el modelo Zero Trust dentro de un API GraphQL limita la capacidad de las empresas

de protegerse contra ataques cibernéticos, por lo cual, es urgente investigar y proponer un enfoque eficiente y efectivo del modelo de seguridad Zero Trust que garantice la protección de la información, así como el perfecto funcionamiento y rendimiento de las APIs GraphQL.

3. Marco teórico referencial

En 2015, Facebook liberó su propio modelado de APIs, conocido como GraphQL, a la comunidad, solucionando el exceso de datos en las respuestas tipo bloque de las APIs ReST, puesto que, una API GraphQL permite obtener solo los datos necesarios, optimizando tiempos de respuesta y el rendimiento de las aplicaciones, muy útil cuando de ahorrar ancho de banda se habla (Pappula & Anasuri, 2021).

Además, las APIs GraphQL son muy utilizadas cuando se tiene múltiples fuentes de datos y sus relaciones independientemente de la complejidad de las mismas. Las APIs GraphQL cuentan con la llamada Federación de GraphQL, la cual permite combinar varios servicios GraphQL u orígenes de datos y mostrarlos en un único grafo, especialmente útil cuando se desea exponer la información de varios microservicios (Haris et al., 2021; Quiña-Mera et al., 2023). Sin embargo, debido a la posibilidad del usuario de recabar tanta información como exponga el API GraphQL, un atacante puede consultar el grafo con toda la información disponible y este intentará devolver una gran cantidad de datos, colapsando el servicio, este ataque es conocido como Ataque de Denegación de Servicios (Denial of Service) (Belhadi et al., 2024).

3.1. Desafíos de Seguridad en GraphQL

Uno de los ataques que genera mayor preocupación al implementar las APIs GraphQL, es precisamente la exposición excesiva de datos (overfetching), lo que puede desencadenar en una respuesta demasiado grande, ocasionando problemas de rendimiento y disponibilidad del aplicativo, a la vez que es un gran fallo en cuanto a seguridad de la información se refiere (Kumar et al., 2024).

Es muy común que en las APIs GraphQL con una implementación estándar y sin mayores seguridades, retornen respuestas extremadamente generosas en

información, ocasionando el colapso de las aplicaciones y desencadenando una auto Denegación de Servicios (DoS) (Lercher et al., 2023).

Todas las vulnerabilidades descritas pudieron evitarse con políticas de seguridad que controlen el acceso, protegiendo los datos de filtraciones. Políticas que eviten la confianza implícita en los usuarios o sistemas que deseen obtener acceso a su información.

3.2. El Modelo de Seguridad Zero Trust

El modelo de seguridad Zero Trust (Confianza Cero), es una arquitectura que retira toda confianza implícita, independientemente del origen del solicitante, en otras palabras, desconfía de usuarios y aplicaciones que se encuentren dentro o fuera de su propia red. De este modo, obliga a sus clientes a identificarse y autenticarse continuamente, permitiendo el acceso a los recursos exclusivamente a los usuarios verificados y validados (Syed et al., 2022).

Dado que los microservicios necesitan acceder y consultarse unos a otros y para facilidad de los desarrolladores y administradores, han dado por válida esta confianza por defecto, pero, al hacerlo aumentaron la superficie de riesgo de ataques cibernéticos.

3.3. Integración de Zero Trust en APIs

En estudios previos, el modelo Zero Trust ya ha sido implementado en los microservicios, obteniendo una mejor seguridad en las interacciones entre recursos, los cuales tienen una mejor distribución, exponiendo en menor medida los datos sensibles (Yeoh et al., 2023).

Para aplicar este modelo a las APIs GraphQL, es necesario contar con políticas de seguridad y acceso granular, todo esto basado en la identidad del usuario y el uso.

Implementar el modelo Zero Trust en las APIs GraphQL, disminuirá el área de ataque, esto gracias a la desconfianza entre recursos y la exigencia de autenticación y autorización continua. Esto también permite que los usuarios y sistemas tengan acceso únicamente a aquellos recursos e información permitidos y no a toda la información, gracias a las políticas de granularidad de la información (Ghasemshirazi et al., 2023).

3.4. Desafíos y Oportunidades

En este estudio se combinará una API GraphQL con el modelo Zero Trust, por lo tanto, se presentarán varios desafíos, como: tener un control continuo en cuanto a autenticación y autorización sumará capas de seguridad y provocará un pequeño retardo en las respuestas del API; aumentará la complejidad en el desarrollo del API, por ende, los costos de desarrollo aumentarán.

Sin embargo, la oportunidad de contar con un mejor y controlado acceso a la información, así como contar con un API con seguridades robustas frente a ciberataques, supera con creces los desafíos y problemas presentados.

3.5. Necesidad de Investigaciones

Previamente se han realizado estudios sobre la implementación del modelo Zero Trust en la arquitectura de microservicios y en APIs (Abgaz et al., 2023; Söylemez et al., 2022), la integración del modelo con APIs GraphQL tiene poca o nula información documentada teórica y práctica, por lo cual, esta investigación pretende proponer un marco referencial tanto en literatura como en su aplicación técnica, para fortalecer las seguridades de las APIs GraphQL sin mermar su rendimiento y eficiencia, a la vez que se reduce su área de ataque.

4. Materiales y metodología

4.1. Diseño de la Investigación

El presente trabajo titulado “Integración de GraphQL y Seguridad Zero Trust en el Diseño de APIs”, se realizará con un enfoque casi experimental, dado que se analizarán libros, estudios, discursos y otros tipos de textos que permitan conocer el funcionamiento de GraphQL y del modelo Zero trust, así como, la medición de tiempos de respuesta, rendimiento y carga, para evaluar la eficacia de la implementación.

4.2. Fases del Estudio

4.2.1. Fase 1: Revisión Bibliográfica y Análisis Teórico

Se revisó la literatura encontrada en bases de datos científicas y técnicas, información relacionada con la implementación de APIs GraphQL, características, beneficios y vulnerabilidades; así como, del modelo de seguridad Zero Trust se investigó sus patrones, arquitecturas sugeridas y buenas prácticas de implementación. Esta revisión permitió identificar las vulnerabilidades en las instalaciones base de las APIs GraphQL y determinar como el modelo Zero Trust puede cubrirlas sin perder rendimiento en la implementación.

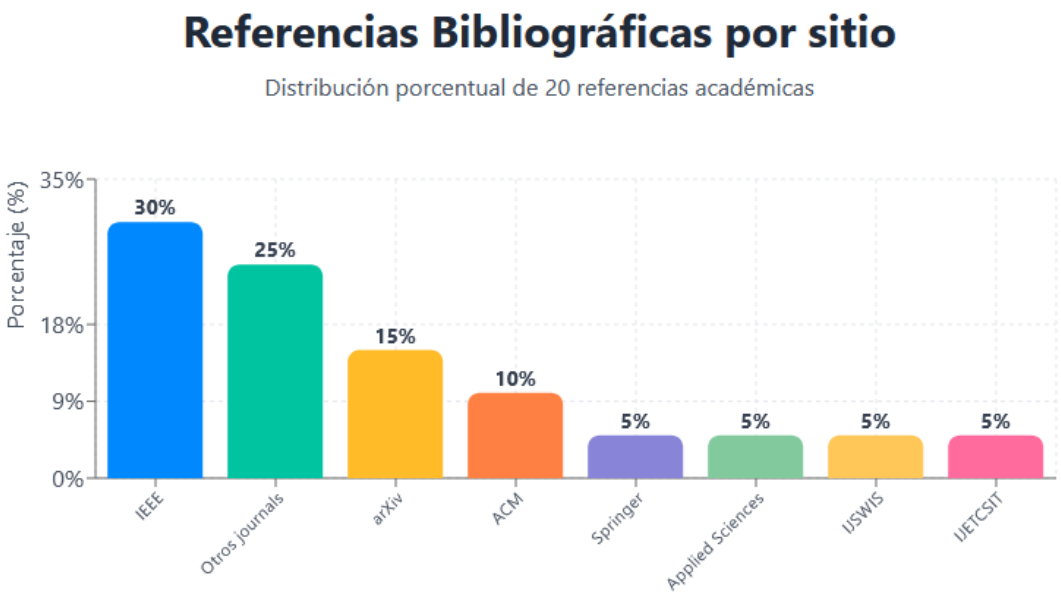

Ilustración 1 Número de referencias por base de datos Científica

Para llevar a cabo la investigación se utilizaron las bases de datos científicas provistas por la Universidad (IEEE Xplore, Scopus, Web of Science, SciELO), así como otras fuentes gratuitas en internet (NIST, Google Scholar) para encontrar documentación científica oficial respecto al tema o partes del mismo.

4.2.2. Fase 2: Análisis y Diseño de la Arquitectura Propuesta

Con base en los hallazgos científicos de la Fase 1, se diseñó un modelo conceptual y una arquitectura técnica detallada de una API GraphQL y el modelo de seguridad Zero Trust.

Se definió:

- Componentes lógicos para la integración.
- Mecanismos para el control de acceso a campos y operaciones en el API GraphQL.
- Herramientas para la autenticación y autorización continuas.

- Métodos para un flujo seguro de la información en el API GraphQL.

4.2.3. Fase 3: Desarrollo del Prototipo Funcional

Se desarrolló una prueba de concepto (PoC) de un API GraphQL base, donde se implementó el modelo de seguridad Zero Trust. Esto, en un ambiente aislado y controlado. La información protegida fue generada automáticamente y de manera dinámica.

Por esto, a continuación, se muestran 3 Diagramas del C4 model:

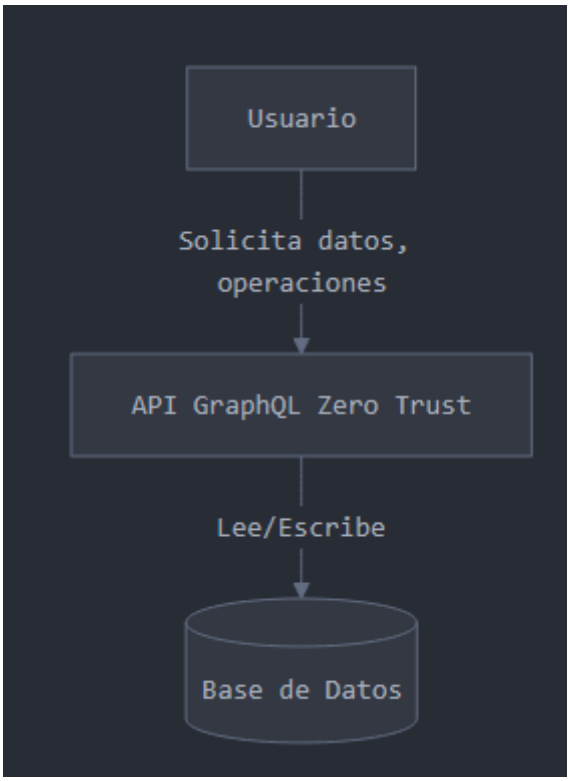

Ilustración 2 Diagrama de contexto

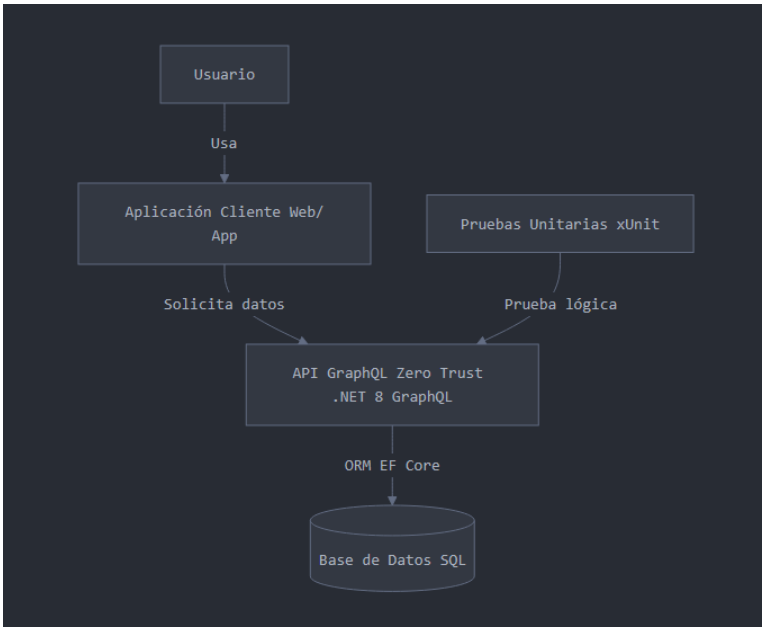

Ilustración 3 Diagrama de contenedores

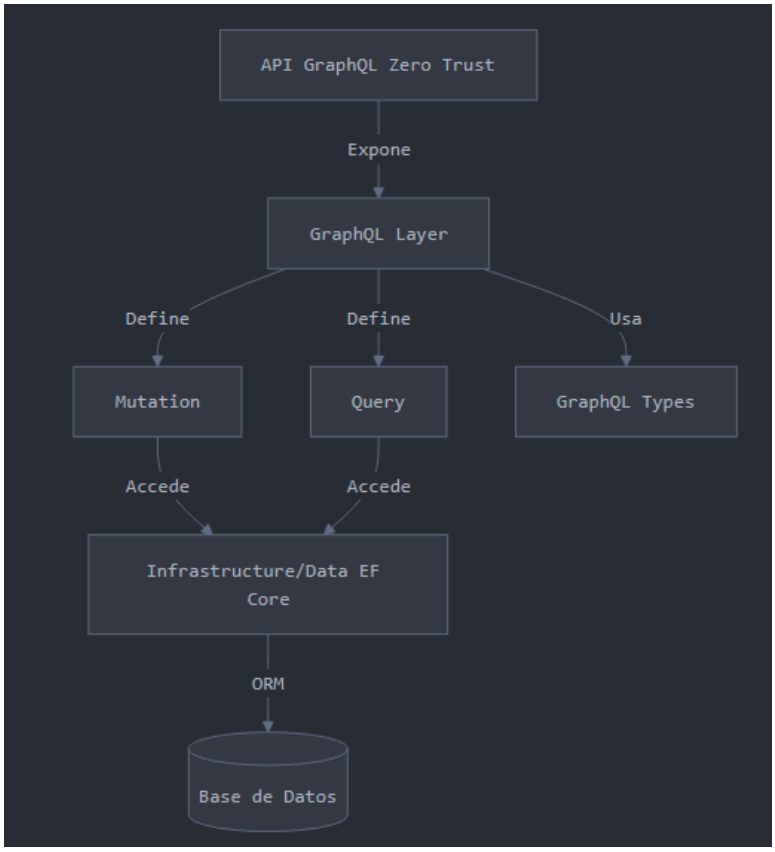

Ilustración 4 Diagrama de componentes

También, se cuenta con el siguiente diagrama donde se pretende indicar la arquitectura con la que contará la PoC:G

Ilustración 5 Arquitectura Técnica

Todas las herramientas y software utilizados son opensource o en su defecto freeware, para evitar el pago de licencias.

Para el desarrollo del prototipo se utilizó .Net 8, C#, Entity Framework Core. De igual manera, se utilizó el IDE Visual Studio 2022 en su distribución Community, misma que es de uso gratuito. El nuget HotChocolate.AspNetCore fue el framework encargado de ayudar en la generación del API GraphQL.

Para la gestión de identidad y autorización se utilizó Keycloak. Luego se usó Vault, de la empresa HashiCorp, para gestionar secretos y credenciales de manera segura. Para asegurar que el contenedor en Docker no tiene fallas de seguridad se usó el plugin Trivy, mismo que escanea el contenedor y emite un informe de haber alguna.

Adicionalmente, y como software adicional, se implementó Seq para monitorear y visualizar logs del sistema.

El sistema gestor de base de datos fue SQL Server 2022 en su distribución Express, misma que es de uso gratuito.

La implementación se realizó enteramente con Docker Desktop, procurando utilizar siempre su última versión estable.

4.2.4. Fase 4: Implementación del prototipo

En cuanto al modelo de datos, se realiza un diseño minimalista mismo que consta de tres tablas: Users, CreditCards y BankAccounts. A continuación, el modelo entidad relación de la base de datos.

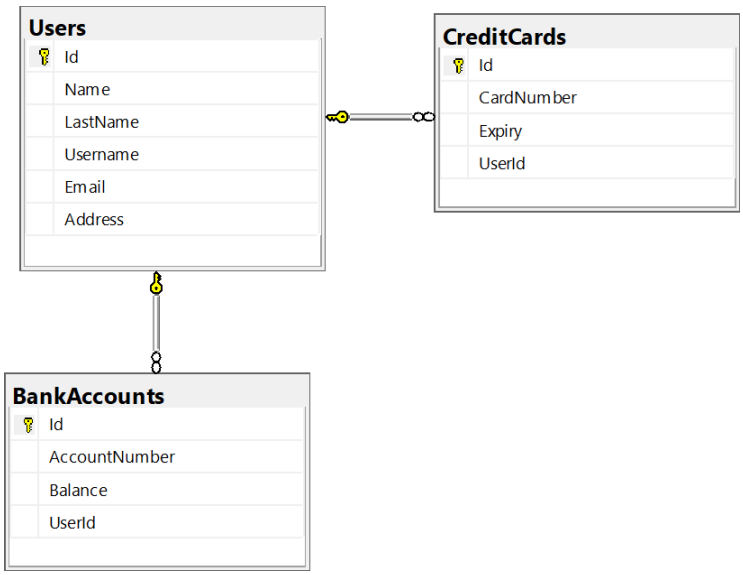

Ilustración 6 Modelo ER base de datos

Para cumplir con el principio de confianza cero, se utilizó el servicio de KeyCloak como servidor de autenticación y autorización. Por tanto, previo a realizar cualquier

consulta a la POC es necesario autenticarse y obtener un token JWT válido del servidor keycloak. Para efectos prácticos se utilizó el software Postman para realizar las peticiones al API ReST de keycloak.

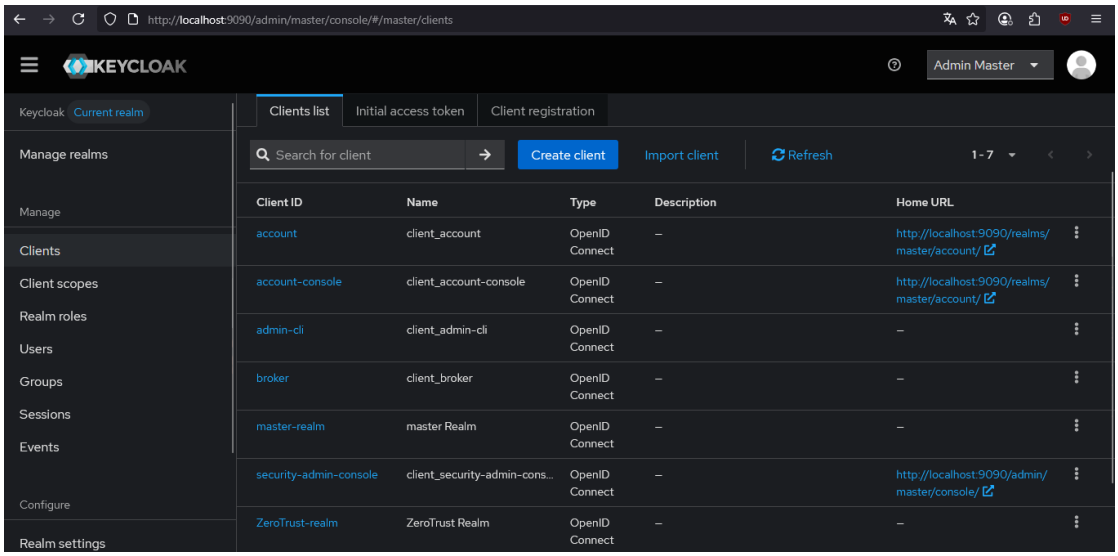

Ilustración 7 Vista principal de KeyCloak

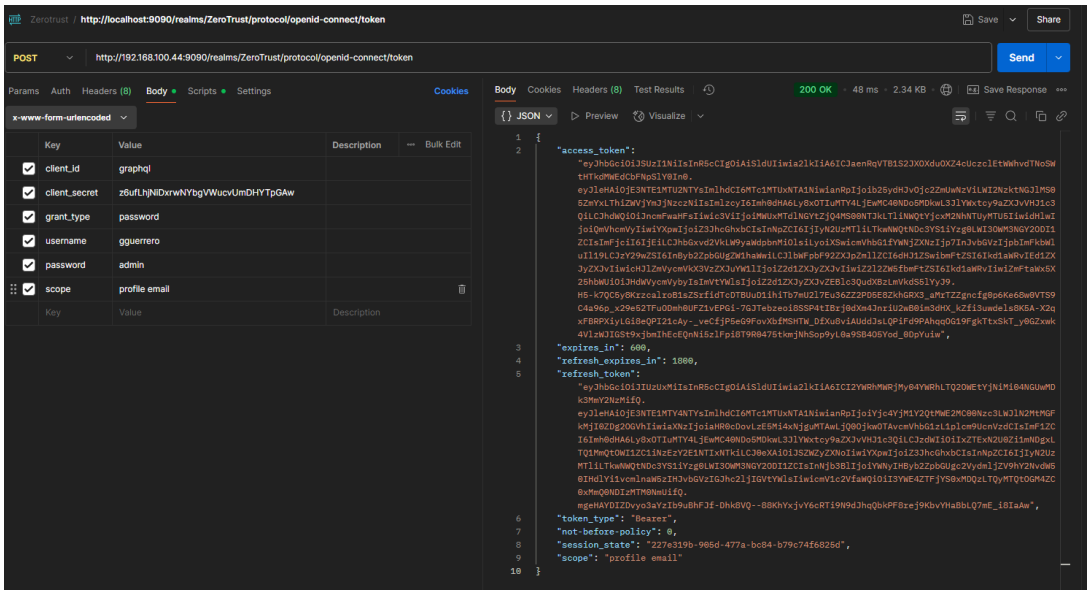

Ilustración 8 Obtención token JWT de keycloak

Por otra parte, el aplicativo no cuenta con un frontend para probar la funcionalidad del API GraphQL, así que, las pruebas se las realizará con Postman.

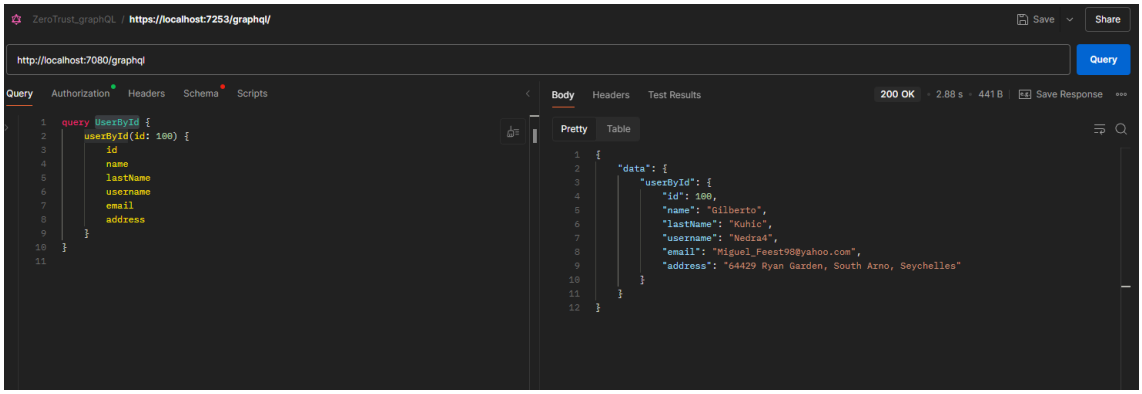

Ilustración 9 Consulta API GraphQL

El backend, donde reside la funcionalidad de la aplicación se realizó en .Net8 y el nuget HotChocolate. A continuación, se agrega fragmentos de código importantes para su correcto funcionamiento.

En el siguiente fragmento se puede observar cómo deshabilitar la introspección en el API GraphQL.

```
using HotChocolate.Resolvers;

namespace GraphQLZeroTrust.Infrastructure.Security
{
    2 referencias
    public class IntrospectionMiddleware
    {
        private readonly FieldDelegate _next;

        0 referencias
        public IntrospectionMiddleware(FieldDelegate next)
        {
            _next = next;
        }

        0 referencias
        public async Task InvokeAsync(IMiddlewareContext context)
        {
            var fieldName = context.Selection.Field.Name;

            if (fieldName.StartsWith("__"))
            {
                context.ReportError("🚫 Introspección deshabilitada por seguridad.");
                return;
            }

            await _next(context);
        }
    }
}
```

Ilustración 10 Configuración para deshabilitar introspección

En el siguiente fragmento se visualiza como se calcula la complejidad de cada consulta, para evitar obtener resultados demasiado grandes, evitando los ataques de tipo DOS.

```
2 referencias
private int CountCost(SelectionSetNode selectionSet)
{
    int cost = 0;

    foreach (var selection in selectionSet.Selections.OfType<FieldNode>())
    {
        int fieldCost = 1; // Valor por defecto si no hay @cost

        var costDirective = selection.Directives.FirstOrDefault(d => d.Name.Value == "cost");
        if (costDirective != null)
        {
            var weightArg = costDirective.Arguments.FirstOrDefault(a => a.Name.Value == "weight");
            if (weightArg?.Value is StringValueNode weightValue &&
                int.TryParse(weightValue.Value, out int parsedWeight))
            {
                fieldCost = parsedWeight;
            }
        }

        cost += fieldCost;

        if (selection.SelectionSet != null)
        {
            cost += CountCost(selection.SelectionSet);
        }
    }

    return cost;
}
```

Ilustración 11 Cálculo de complejidad de consultas

El fragmento a continuación está relacionado con el manejo de secretos (información sensible) que no puede encontrarse dentro de la aplicación y como obtenerlos desde el servidor Vault by Hashicorp:

```
public class VaultService : IVaultService
{
    private readonly IVaultClient _vaultClient;

    0 referencias
    public VaultService(IConfiguration configuration)
    {
        var vaultAddress = configuration["Vault:Address"];
        var vaultToken = configuration["Vault:Token"];

        var authMethod = new TokenAuthMethodInfo(vaultToken);
        var vaultClientSettings = new VaultClientSettings(vaultAddress, authMethod);
        _vaultClient = new VaultClient(vaultClientSettings);
    }

    5 referencias
    public async Task<string> GetSecretAsync(string path, string key)
    {
        Secret<SecretData> secret = await _vaultClient.V1.Secrets.KeyValue.V2.ReadSecretAsync(path, mountPoint: "kv");
        return secret.Data.Data[key]?.ToString() ?? throw new Exception($"Key '{key}' not found in Vault.");
    }
}
```

Ilustración 12 Obtener secretos desde Vault

En contrapartida, se tiene el servidor Vault donde residen dichos secretos:

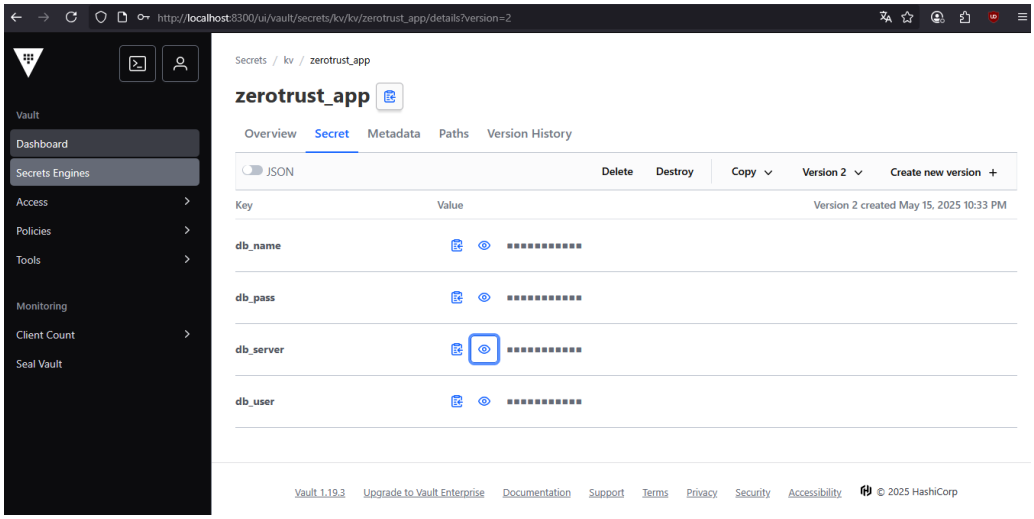

Ilustración 13 Vista principal servidor administrador de secretos Vault

En el desarrollo de software es necesario realizar pruebas unitarias para validar que el código agregado funciona de la manera esperada. Un ejemplo de prueba unitaria es:

```
using GraphQLZeroTrust.API.GraphQL;
using GraphQLZeroTrust.API.GraphQL.Types;
using GraphQLZeroTrust.Infrastructure.Data;
using Microsoft.EntityFrameworkCore;

namespace GraphQLZeroTrustTest.API
{
    public class MutationTests
    {
        [Fact]
        public async Task CreateUserAsync_ShouldAddUserToDatabase()
        {
            // Arrange
            var options = new DbContextOptionsBuilder<AppDbContext>()
                .UseInMemoryDatabase(databaseName: "CreateUserTestDB")
                .Options;

            var input = new CreateUserInput(
                "Juan",
                "Pérez",
                "jperez",
                "juan.perez@example.com",
                "Quito"
            );

            using var context = new AppDbContext(options);
            var mutation = new Mutation();

            // Act
            var result = await mutation.CreateUserAsync(context, input);

            // Assert
            Assert.NotNull(result);
            Assert.Equal("Juan", result.Name);
            Assert.Equal(1, await context.Users.CountAsync());
        }

        [Fact]
        public async Task CreateUserAsync_ShouldThrow_WhenSaveFails()
        {
            // Arrange
            var options = new DbContextOptionsBuilder<AppDbContext>()
                .UseInMemoryDatabase("FailOnSaveTest")
                .Options;

            var input = new CreateUserInput(
                "Juan",
                "Pérez",
                "jperez",
                "juan.perez@example.com",
                "Quito"
            );

            var context = new AppDbContext(options);
            var mutation = new Mutation();

            await mutation.CreateUserAsync(context, input);

            // Assert
            Assert.Equal(1, await context.Users.CountAsync());
        }
    }
}
```

Ilustración 14 Unit Tests de Mutations

4.2.5. Fase 5: Pruebas y Validación

Con la finalidad de evaluar la eficiencia y seguridad de la implementación, se ejecutaron pruebas de rendimiento, seguridad y funcionales, tanto sobre el prototipo base, como sobre el API GraphQL con seguridad del modelo Zero Trust.

Estas pruebas midieron los tiempos de respuesta cuando el API reciba conjuntos definidos de peticiones.

En la ilustración siguiente se puede observar los resultados de la prueba de carga realizada con el plugin Artillery (para NodeJs) al API GraphQL sin seguridades. Esta prueba envió peticiones de 300 usuarios, donde el valor principal lo tiene la cantidad de errores 401, es decir, error de autorización, mismo que tiene un valor de cero (0), puesto que no cuenta con ningún filtro de acceso de usuarios.

Contadores Agregados

Concepto	Valor
Usuarios creados	300
Solicitudes HTTP	300
Errores 400 (Solicitud Incorrecta)	18
Respuestas HTTP	300
Bytes descargados	28951
Usuarios fallidos	0
Usuarios completados	300
Errores 429 en GraphQL	50
Errores 401 en GraphQL	0

Ilustración 15 Contadores de peticiones GraphQL sin seguridades

Una vez agregadas las seguridades del modelo Zero Trust, se realizó la misma prueba con 300 usuarios y se pudo observar que la cantidad de errores 401 subió a

40, y, es por ello que la cantidad de bytes descargados disminuyó, lo cual también sirve para aliviar la carga de procesamiento del servidor.

Contadores Agregados

Concepto	Valor
Usuarios creados	300
Solicitudes HTTP	300
Errores 400 (Solicitud Incorrecta)	10
Respuestas HTTP	300
Bytes descargados	16330
Usuarios fallidos	0
Usuarios completados	300
Errores 429 en GraphQL	250
Errores 401 en GraphQL	40

Ilustración 16 Contadores de peticiones GraphQL con seguridades

Durante la Fase 1 se encontraron diferentes vulnerabilidades de las API GraphQL, por tanto, los cuales fueron reproducidos sobre el API. También, se validó que los controles de acceso, la autenticación y autorización continua funcionen de la manera esperada. Esto, para medir la robustez de la implementación.

Uno de los problemas de seguridad es la introspección, mismo que fue deshabilitado y puesto a prueba, en la siguiente ilustración se puede observar el resultado

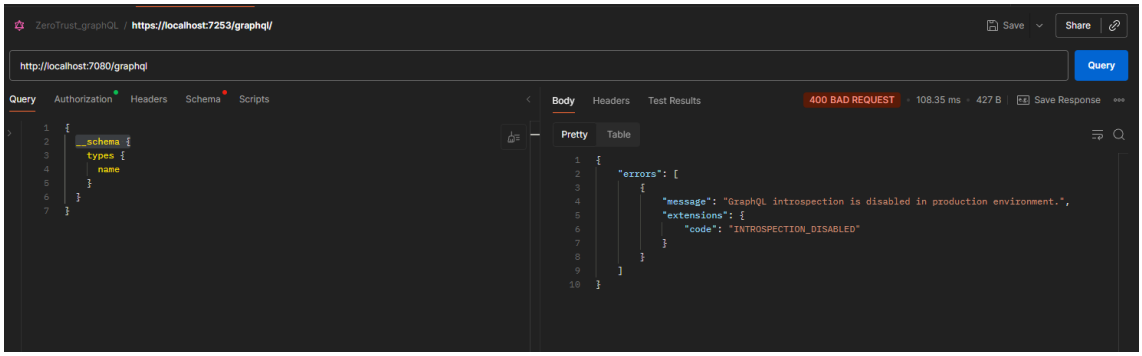

Ilustración 17 Introspección deshabilitada

5. Resultados y discusión

5.1. Resultados

Cada vez que un usuario necesitaba consultar o consumir los servicios del API GraphQL requería autenticación y autorización previa, por lo cual se utilizó keycloak como proveedor de identidades y servidor de autorización mediante tokens JWT (Json Web Token).

De igual manera, se establecieron configuraciones en el código fuente del aplicativo para desactivar la introspección, misma que permite que un usuario no autenticado y autorizado pueda obtener el esquema completo del API GraphQL.

Si un usuario previamente autenticado y autorizado intenta obtener una gran cantidad de información de una API GraphQL, es posible que al API sufra un ataque del tipo DOS (Denial of Service), por tanto, para evitar esto se han aplicado, directamente en la implementación del API, límites de profundidad y complejidad a las consultas.

Estas validaciones fueron realizadas previo a las pruebas, para garantizar que el API GraphQL es segura de acuerdo al modelo de Seguridad Zero Trust.

Se simuló el uso del API GraphQL por 300 usuarios concurrentes, donde cada usuario realizó consultas con las políticas de seguridad activas. En la Ilustración 16 se puede ver los resultados de estas.

En los resultados, se puede observar que de las 300 solicitudes se pudo gestionar el 100% de las conexiones concurrentes sin fallos o interrupciones fatales.

En cuanto a los errores 429 (Too Many Requests), se encontró que 250 usuarios intentaron realizar consultas excesivamente costosas o que superaban los límites

de complejidad, demostrando así que el sistema se encuentra protegido y las restricciones funcionan de una manera adecuada.

Los errores 401 indican que el token de 40 peticiones no era válido, por tanto, el sistema respondió de la manera esperada, es decir, negando el acceso a estas peticiones, cumpliendo estrictamente el modelo de Seguridad Zero Trust.

Finalmente, los errores 400 (Bad Request), son errores donde la petición al API GraphQL estuvo mal formada, un riesgo presente en las pruebas automatizadas.

Resumen Tiempo de Respuesta HTTP (ms)

Tiempo Mínimo (ms)	Tiempo Máximo (ms)	Cantidad de Requests	mean	p50	Mediana (ms)	p75	p90	Percentil 95 (ms)	Percentil 99 (ms)	p999
1	19	300	1.8	2	2	2	2	3	4	5

Ilustración 18 Resumen tiempos de respuesta

En cuanto al Resumen Tiempo de Respuesta HTTP (ms), se puede observar que la mediana es de apenas 2 milisegundos, lo cual indica que pese a las seguridades y validaciones en cada petición el API GraphQL es altamente eficiente.

Para unos pocos casos, el tiempo máximo de respuesta fue de 19 milisegundos, sin embargo, al constatar que la mayoría de los percentiles se encuentran dentro del rango de 2 a 4 milisegundos, es posible decir que estos son casos aislados muy probablemente por demoras de red en la validación del token o causados por proceso de simulación.

Adicionalmente, se realizó un escaneo de vulnerabilidades con Trivy al contenedor de la aplicación, donde el resultado fue cero vulnerabilidades.

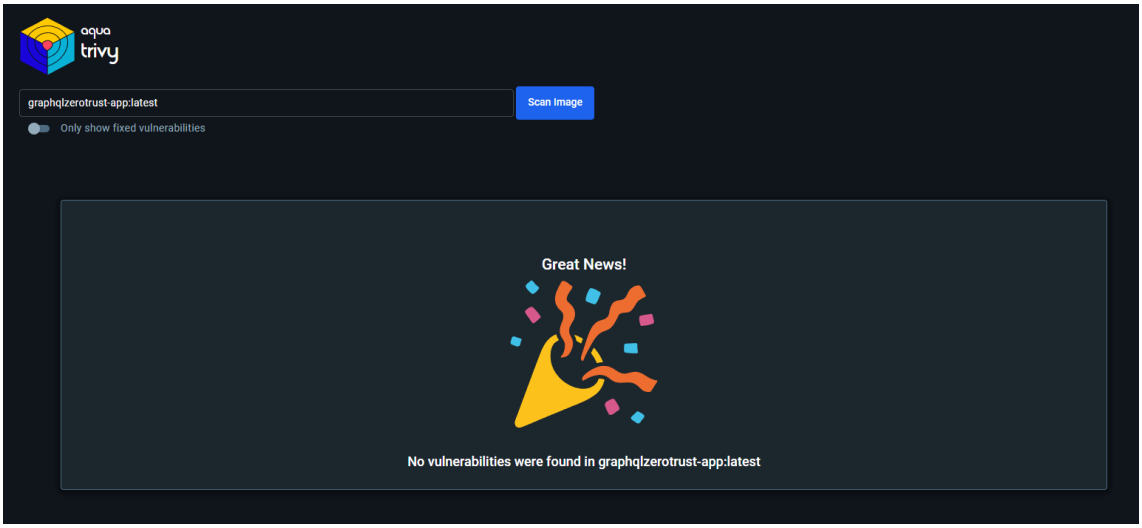

Ilustración 19 Escaneo de vulnerabilidades con Trivy

5.2. Discusión

En el lapso de desarrollo e implementación del prototipo se obtuvo hallazgos relacionados a las ideas encontradas en la bibliografía, entre ellos, la facilidad y flexibilidad que posee el API GraphQL para modelar las respuestas a consultas con respuestas de pequeño o gran tamaño. Sin embargo, esta facilidad puede ser utilizada para la introspección y/o ataques DOS, algo que ya fue señalado por Belhadi, Zhang y Arcuri (2024). Por ello, en el presente proyecto, como parte de la seguridad del modelo Zero Trust, se implementaron estrategias y controles para deshabilitar o controlar la introspección y el costo de las consultas, puesto que de no hacerlo la carga del servidor aumentó a niveles donde la aplicación dejó de funcionar.

Igualmente, se ha seguido las recomendaciones de Syed et al. (2022) y Dhiman et al. (2024), quienes dicen que la autenticación, validación de usuarios y el principio del mínimo privilegio debe formar parte del ciclo de vida de cada consulta, por lo tanto, se han implementado reglas de identidad y acceso a los recursos del API GraphQL, impidiendo que usuarios no autorizados que hagan consultas simples tengan acceso a los datos sensibles como las cuentas bancarias o números de tarjetas de crédito. Este tipo de accesos lo tiene únicamente los usuarios con rol,

Admin y el dueño de cada información; verificando el enfoque de Kang, Liu, Wang, Meng y Liu (2023), quienes aseguran que el modelo de seguridad Zero Trust no solo es una capa adicional, sino que se trata de un componente esencial en la aplicación.

Rodigari, O'Shea, McCarthy, McCarry y McSweeney (2021), sostienen que una correcta implementación del modelo de seguridad Zero Trust no debe comprometer el rendimiento de una aplicación segura, y, sin duda tienen razón ya que los resultados de las pruebas de carga del API GraphQl revelan que las respuestas se mantienen dentro de márgenes aceptable, es decir, menos de 19 milisegundos en el caso de mayor tiempo de respuesta cuando el margen aceptable es de 300 milisegundos. Es necesario indicar que, para obtener la respuesta se pasa por varias capas de seguridad y monitores: verificación de token, control de políticas, auditoría de logs y control de campos.

Lercher, Glock, Macho y Pinzger (2023) es su escrito argumentan que migrar de API ReST a GraphQL requiere una curva de aprendizaje referente a la seguridad y sus controles asociados, lo cual es muy cierto, puesto que fue necesario aplicar políticas de seguridad manualmente. Además, que estas medidas de seguridad permiten retornar volúmenes de datos controlados, alrededor del 94% menos de campos retornados comparados con APIs ReST, optimizando el consumo de red y tiempo de espera del usuario, coincidiendo con los hallazgos de Pappula y Anasuri (2021), quienes en su estudio lograron evidenciar que un API GraphQL correctamente implementada tiene mayor eficiencia frente a APIs ReST.

Existen amenazas que no se encuentra directamente en la implementación, sino en sus contenedores, Trivy y Docker Bench for Security permitieron analizar, encontrar y mitigar las posibles amenazas en los contenedores utilizados, ideas que Kumar S., Dwivedi, Kumar M. y Gill (2024) apoyan manifestando que la seguridad de una aplicación no se encuentra solo en su código, sino también del entorno donde se encuentra alojada. En el estudio realizado por Singh y Gupta (2022), se destaca la importancia de aplicar varias capas de seguridad en la implementación las aplicaciones actuales y no solo en el código fuente.

Por otra parte, se tiene la seguridad reactiva, puesto que como indica Aslan et al (2023), parte de la seguridad es supervisar y analizar los registros en busca de ataques, accesos no autorizados o intentos de explotación de vulnerabilidades, mismos que deben ser corregidos a la brevedad posible; y, para esto se implementó Serilog y Seq, para generar y recoger los logs de la aplicación y su funcionamiento.

Finalmente, herramientas como Keycloak y Vault by HashiCorp, permitieron desarrollar una solución que según el trabajo de AlDaajeh y Alrabaee (2024) debe ser segura, flexible, adaptable y económicamente viable, logrando que pueda ser implementada tanto en entornos comerciales, gubernamentales como catedráticos, proponiendo el modelo de seguridad Zero Trust como una estrategia sólida y adaptable, una idea que la reconocen Ghasemshirazi, Shirvani y Alipour (2023).

Con toda la evidencia obtenida en la etapa de implementación, es posible decir que la prueba de concepto ha sido un éxito, ya que no solo demuestra que integrar un API GraphQL y el modelo de seguridad Zero Trust es posible, sino que van acorde a las ideas de la literatura académica revisada, validando que combinar las dos tecnologías permiten construir APIs GraphQL eficientes, auditables y seguras desde su diseño.

6. Conclusiones y trabajo futuro

Mediante el presenta trabajo de titulación se ha demostrado que integrar el modelo de Seguridad Zero Trust en un API GraphQL sin impactar en su rendimiento es posible.

La implementación básica del API GraphQL contiene varios riegos de seguridad, como overfetching, GraphQL Injection, introspección excesiva y denegación de servicio; mismos que suelen ser explotados por atacantes en busca de información confidencial. Todos estos riesgos pudieron ser mitigados quitando la confianza implícita, autenticando cada petición, a la vez que se evalúa la complejidad de cada consulta, evitando la consulta de grandes cantidades de información y verificando que la información sensible solo está disponible para los usuarios con el rol o permisos correctos. En otras palabras, el modelo de Seguridad Zero Trust eliminó los riesgos indicados.

La arquitectura propuesta implementó un autenticador y autorizador de identidades (keycloak), la información confidencial, conocida como secretos se encuentran alojados en un servidor dedicado llamado Vault desarrollado por Hashicorp. El control y verificación de registros de auditoria es esencial, para lo cual se implementó una combinación de Serilog y Seq, para la visualización y filtrado de manera amistosa para el usuario. Para finalizar, se utilizó el plugin Trivy para escanear el contenedor como una estrategia que complementa la seguridad.

La simulación de pruebas ha demostrado que, pese a tener capas de seguridad, el sistema mantiene su eficiencia, estabilidad y fiabilidad. Gracias a los datos sintéticos guardados en la base de datos, se pudo medir los tiempos de respuesta, todo de manera segura y aislada.

Finalmente, el presente estudio es una contribución replicable y concreta que permitirá a las organizaciones implementar el modelo de Seguridad Zero Trust

desde el diseño del API GraphQL, y a futuro validar la arquitectura desarrollada en un entorno productivo que utilice datos reales, donde será posible evaluar el comportamiento del modelo de Seguridad Zero Trust integrado en un API GraphQL en escenarios operativos más complejos, donde intervienen múltiples usuarios, servicios e información de distinta sensibilidad y escala, demostrando así, que la combinación de eficiencia y seguridad es posible.

Referencias

Abgaz, Y., McCarren, A., Elger, P., Solan, D., Lapuz, N., Bivol, M., Jackson, G., Yilmaz, M., Buckley, J., & Clarke, P. (2023). Decomposition of monolith applications into microservices architectures: A systematic review. IEEE Transactions on Software Engineering, 49(8), 4213--4242. https://doi.org/10.1109/TSE.2023.3287297

AlDaajeh, S., & Alrabaee, S. (2024). Strategic cybersecurity. Computers & Security, 141, 103845. https://doi.org/10.1016/j.cose.2024.103845

Aslan, Ö., Aktuğ, S. S., Ozkan-Okay, M., Yilmaz, A. A., & Akin, E. (2023). A Comprehensive Review of Cyber Security Vulnerabilities, Threats, Attacks, and Solutions. Electronics, 12(6), 1333. https://doi.org/10.3390/electronics12061333

Belhadi, A., Zhang, M., & Arcuri, A. (2024). Random testing and evolutionary testing for fuzzing GraphQL APIs. ACM Transactions on the Web, 18(1), 14:1--14:41. https://doi.org/10.1145/3609427

Dhiman, P., Saini, N., Gulzar, Y., Turaev, S., Kaur, A., Nisa, K. U., & Hamid, Y. (2024). A review and comparative analysis of relevant approaches of zero trust network model. Sensors, 24(4), 1328. https://doi.org/10.3390/s24041328

Ghasemshirazi, S., Shirvani, G., & Alipour, M. A. (2023). Zero trust: Applications, challenges, and opportunities. arXiv. https://doi.org/10.48550/arXiv.2309.03582

Haris, M., Farfar, K. E., Stocker, M., & Auer, S. (2021, November). Federating scholarly infrastructures with GraphQL. In International Conference on Asian Digital Libraries (pp. 308-324). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-91669-5_24

Kang, H., Liu, G., Wang, Q., Meng, L., & Liu, J. (2023). Theory and Application of Zero Trust Security: A Brief Survey. Entropy, 25(12), 1595. https://doi.org/10.3390/e25121595

Kumar, S., Dwivedi, M., Kumar, M., & Gill, S. S. (2024). A comprehensive review of vulnerabilities and AI-enabled defense against DDoS attacks for securing cloud services. Computer science review, 53, 100661. https://doi.org/10.1016/j.cosrev.2024.100661

Lercher, A., Glock, J., Macho, C., & Pinzger, M. (2023). Microservice API evolution in practice: A study on strategies and challenges. arXiv preprint arXiv:2311.08175. https://doi.org/10.48550/arXiv.2311.08175

Liu, W., Lin, Y., Qi, Z., Wu, Z., Zhang, G., Wang, K., ... & Yang, Z. (2021, October). LoginSoEasy: a System Enabling both Authentication and Protection of Personal Information based on Trusted User Agent. In 2021 IEEE Sixth International Conference on Data Science in Cyberspace (DSC) (pp. 122-129). IEEE. https://doi.org/10.1109/DSC53577.2021.00024

Pappula, K. K., & Anasuri, S. (2021). API Composition at Scale: GraphQL Federation vs. REST Aggregation. International Journal of Emerging Trends in Computer Science and Information Technology, 2(2), 54-64. https://doi.org/10.63282/3050-9246.IJETCSIT-V2I2P107

Quiña-Mera, A., Fernandez, P., García, J. M., & Ruiz-Cortés, A. (2023). GraphQL: A systematic mapping study. ACM Computing Surveys, 55(10). https://doi.org/10.1145/3561818

Rodigari, S., O'Shea, D., McCarthy, P., McCarry, M., & McSweeney, S. (2021). Performance analysis of Zero-Trust multi-cloud. In 2021 IEEE 14th International Conference on Cloud Computing (CLOUD) (pp. 730--732). IEEE. https://doi.org/10.1109/CLOUD53861.2021.00097

Shah, D., Shah, D., Hegdepatil, P., & Toggi, A. (2024). GraphQL---Time for some introspection. Proceedings of the 2024 13th International Conference on System Modeling & Advancement in Research Trends (SMART), 649--654. https://doi.org/10.1109/SMART63812.2024.10882558

Denzel, K. (2025). A survey of security in zero trust network architectures. https://doi.org/10.30574/gscarr.2025.22.2.0036

Singh, A. & Gupta, B. B. (2022). Distributed Denial-of-Service (DDoS) Attacks and Defense Mechanisms in Various Web-Enabled Computing Platforms: Issues, Challenges, and Future Research Directions. International Journal on Semantic Web and Information Systems (IJSWIS), 18(1), 1-43. https://doi.org/10.4018/IJSWIS.297143

Söylemez, M., Tekinerdogan, B., & Kolukısa Tarhan, A. (2022). Challenges and Solution Directions of Microservice Architectures: A Systematic Literature Review. Applied Sciences, 12(11), 5507. https://doi.org/10.3390/app12115507

Syed, N. F., Shah, S. W., Shaghaghi, A., Anwar, A., Baig, Z., & Doss, R. (2022). Zero trust architecture (ZTA): A comprehensive survey. IEEE Access, 10, 57143--57179. https://doi.org/10.1109/ACCESS.2022.3174679

Yeoh, W., Liu, M., Shore, M., & Jiang, F. (2023). Zero trust cybersecurity: Critical success factors and A maturity assessment framework. Computers & Security, 133, 103412. https://doi.org/10.1016/j.cose.2023.103412