Journal of Information System Research (JOSH)
Volume 7, No. 4, Juli 2026, pp 934−943
ISSN 2686-228X (media online)
https://ejurnal.seminar-id.com/index.php/josh/
DOI 10.47065/josh.v7i4.9874

Kuantifikasi Risiko Introspection pada Tiga Kategori Otorisasi OWASP: Studi Komparatif REST API dan GraphQL

Naufal Hanif Athallah[1,*], Galet Guntoro Setiaji[1], Ahmad Rifa'i[2]

[1]Fakultas Teknologi Informasi dan Komunikasi, Program Studi Teknik Informatika, Universitas Semarang, Semarang
Jl. Soekarno Hatta, Tlogosari Kulon, Kec. Pedurungan, Kota Semarang, Jawa Tengah, Indonesia
[2]Fakultas Teknologi Informasi dan Komunikasi, Program Studi Sistem Informasi, Universitas Semarang, Semarang
Jl. Soekarno Hatta, Tlogosari Kulon, Kec. Pedurungan, Kota Semarang, Jawa Tengah, Indonesia
Email: [1,*]naufalhanifath125@gmail.com, [2]gallet@usm.ac.id, [3]rifai@usm.ac.id
Email Penulis Korespondensi: naufalhanifath125@gmail.com
Submitted: **07/05/2026**; Accepted: **04/07/2026**; Published: **05/07/2026**

Abstrak−Perkembangan Application Programming Interface (API) menuntut evaluasi keamanan tingkat arsitektural secara terukur. Penelitian ini mengkuantifikasi risiko keamanan REST API dan GraphQL berdasarkan tiga kategori otorisasi dari OWASP API Security Top 10 2023 (API1, API3, dan API5). Pembatasan eksklusif pada ketiga kategori ini dilakukan karena fokus riset murni mengevaluasi cacat logika kontrol akses, bukan kerentanan tingkat infrastruktur. Pengujian dilakukan menggunakan TixVuln, instrumen testbed arsitektur paralel yang dirancang khusus untuk mengeliminasi bias basis data eksternal sebuah keunggulan komparatif yang tidak dimiliki oleh aplikasi rentan tunggal standar industri. Evaluasi otorisasi dieksekusi secara berkonteks untuk menghindari tingginya false-negative yang dihasilkan oleh instrumen pemindaian keamanan otomatis (SAST/DAST) pada pengujian logika bisnis. Hasil kuantifikasi menggunakan OWASP Risk Rating Methodology mengungkap kebaruan (novelty) bahwa GraphQL mengalami eskalasi kategori risiko dari tingkat Sedang menjadi Kritis pada API3 dan API5 dibandingkan REST API. Lompatan signifikansi metrik kemudahan penemuan (Ease of Discovery) ini dipicu secara absolut oleh eksposur skema operasional melalui fitur introspection. Pengujian versi mitigasi memvalidasi bahwa penerapan field whitelisting dan Role-Based Access Control tingkat resolver mutlak diperlukan untuk menekan risiko inheren pada arsitektur single-endpoint. Kontribusi utama penelitian ini adalah penyediaan kerangka evaluasi empiris terisolasi yang membuktikan secara kuantitatif bahwa fleksibilitas arsitektur GraphQL berbanding lurus dengan peningkatan fatalitas risiko otorisasi apabila fitur schema discovery tidak dikonfigurasi secara ketat.

Kata Kunci: GraphQL; Introspection; Kuantifikasi Risiko; OWASP API Security Top 10 2023; REST API; Testbed Keamanan.

Abstract−The advancement of Application Programming Interfaces (APIs) demands measurable architectural-level security evaluation. This study quantifies the security risks of REST API and GraphQL based on three authorization categories from the OWASP API Security Top 10 2023 (API1, API3, and API5). The exclusive limitation to these three categories was established to focus purely on access control logic flaws rather than infrastructure-level vulnerabilities. The experiment utilizes TixVuln, a parallel-architecture testbed instrument explicitly designed to eliminate external database bias a comparative advantage not present in standard single-architecture vulnerable applications. Authorization evaluation was executed contextually to avoid the high false-negative rates typically produced by automated security scanning tools (SAST/DAST) in business logic testing. Quantification results using the OWASP Risk Rating Methodology reveal a novelty that GraphQL experiences a risk category escalation from Medium to Critical levels in API3 and API5 compared to REST API. This significant leap in the Ease of Discovery metric is absolutely triggered by the operational schema exposure through the introspection feature. Mitigation testing validates that implementing field whitelisting and resolver-level Role-Based Access Control is imperative to suppress inherent risks in single-endpoint architectures. The main contribution of this research is the provision of an isolated empirical evaluation framework that quantitatively proves the flexibility of GraphQL architecture is directly proportional to the increased fatality of authorization risks if the schema discovery feature is not strictly configured.

Keywords: API Security; GraphQL; Introspection; OWASP API Security Top 10 2023; REST API; Risk Quantification.

1. PENDAHULUAN

Arsitektur perangkat lunak kontemporer saat ini menempatkan Application Programming Interface (API) sebagai fondasi utama dalam menjembatani komunikasi antar-komponen sistem yang terdistribusi [1], [2]. Dalam ekosistem digital yang terus berkembang, API tidak lagi sekadar antarmuka teknis, melainkan telah bertransformasi menjadi aset strategis yang menentukan keandalan, skalabilitas, dan keamanan sebuah sistem informasi secara keseluruhan. Di antara berbagai standar yang tersedia, Representational State Transfer (REST) API dan GraphQL muncul sebagai dua paradigma yang paling dominan di industri pengembangan perangkat lunak modern [3], [4], [5]. Meskipun REST telah mengukuhkan posisinya sebagai standar de facto selama lebih dari dua dekade karena kemudahan implementasi dan dukungan ekosistem yang matang, kehadiran GraphQL yang diinisiasi oleh Facebook pada 2015 menawarkan keunggulan berupa fleksibilitas query yang lebih dinamis serta efisiensi transfer data yang lebih presisi [6], [7].

Namun, perbedaan fundamental antara keduanya tidak semata terletak pada pola routing, melainkan pada mekanisme parsing dan execution query yang lebih dalam. REST API memproses setiap permintaan secara statis berdasarkan rute URL yang telah didefinisikan, sehingga struktur endpoint bersifat implisit dan hanya dapat diketahui melalui dokumentasi eksternal. Sebaliknya, GraphQL menggunakan mekanisme parsing dan execution yang terpusat pada satu endpoint serta menyediakan fitur introspection yang memungkinkan klien memperoleh

Journal of Information System Research (JOSH)
Volume 7, No. 4, Juli 2026, pp 934−943
ISSN 2686-228X (media online)
https://ejurnal.seminar-id.com/index.php/josh/
DOI 10.47065/josh.v7i4.9874

informasi mengenai struktur skema (schema) API secara dinamis. Kombinasi antara mekanisme parsing terpusat dan kemampuan schema discovery tersebut berpotensi meningkatkan kemudahan penemuan (Ease of Discovery) terhadap objek maupun fungsi yang dapat menjadi target eksploitasi otorisasi. Perbedaan pada lapisan execution inilah yang berpotensi melahirkan profil risiko otorisasi yang berbeda secara signifikan antara keduanya, khususnya dalam konteks kemudahan penemuan (ease of discovery) terhadap objek data dan fungsi administratif [8], [9].

Eskalasi ancaman terhadap integritas API kini menjadi perhatian serius di level global, seiring dengan meningkatnya ketergantungan organisasi terhadap layanan berbasis API dalam operasional bisnis sehari-hari [10]. Fenomena ini kemudian direspons oleh OWASP (Open Web Application Security Project) melalui peluncuran OWASP API Security Top 10 2023 sebagai revisi komprehensif atas versi 2019, yang memetakan sepuluh kategori kerentanan paling krusial pada level API berdasarkan data insiden keamanan terkini dari komunitas praktisi global [11], [12]. Kerangka kerja ini menjadi rujukan utama bagi komunitas keamanan siber dalam mengidentifikasi, mengevaluasi, dan memitigasi risiko pada implementasi API di berbagai industri.

Dari sepuluh kategori kerentanan yang tercantum dalam OWASP API Security Top 10 2023, tiga kategori yang berkaitan dengan mekanisme otorisasi menjadi sorotan utama dalam penelitian ini. Kategori API1 (Broken Object Level Authorization/BOLA) mengacu pada kondisi di mana sistem gagal memvalidasi apakah pengguna yang melakukan request memiliki hak akses terhadap objek data yang diminta, sehingga penyerang dapat mengakses data milik pengguna lain hanya dengan memanipulasi nilai identifikasi objek. Kategori API3 (Broken Object Property Level Authorization) atau yang dikenal sebagai Mass Assignment, merujuk pada kerentanan di mana sistem menerima dan memproses seluruh atribut yang dikirimkan oleh klien tanpa melakukan validasi pembatasan field, sehingga penyerang dapat memodifikasi properti sensitif seperti role atau saldo akun yang seharusnya tidak dapat diubah oleh pengguna biasa. Sementara itu, kategori API5 (Broken Function Level Authorization/BFLA) terjadi ketika sistem tidak memverifikasi apakah pengguna memiliki hak akses fungsional untuk menjalankan operasi tertentu, khususnya operasi administratif yang seharusnya dibatasi hanya untuk peran tertentu [13]. Ketiga kategori ini dipilih karena memiliki keterkaitan erat pada dimensi otorisasi dan secara konsisten muncul sebagai kerentanan yang paling banyak ditemukan pada audit keamanan API di berbagai platform. Ketiga kategori ini dipilih secara eksklusif karena memiliki akar kerentanan pada cacat logika kontrol akses tingkat bisnis (business logic flaws), berbeda dengan kategori lain seperti API4 (Unrestricted Resource Consumption) yang lebih berfokus pada eksploitasi limitasi sumber daya infrastruktur jaringan.

Meskipun berbagai penelitian telah membahas keamanan REST API maupun GraphQL, sebagian besar masih berfokus pada aspek tertentu sehingga belum memberikan evaluasi komparatif yang komprehensif. Napisa et al. [14] mengidentifikasi kerentanan information disclosure pada GraphQL melalui eksploitasi fitur introspection, sedangkan Santos Filho et al. [15] mengembangkan mekanisme deteksi Broken Object Level Authorization (BOLA) pada REST API menggunakan transformasi OpenAPI ke Colored Petri Nets. Di sisi lain, Mazidi et al. [16] mengkaji kerentanan mass assignment pada REST API tanpa mempertimbangkan karakteristik arsitektural GraphQL sebagai faktor pembeda. Sementara itu, penelitian oleh Pratama et al. [17] serta Sadaf et al. [18] lebih menitikberatkan pada perbandingan performa, autentikasi, dan skalabilitas antara REST API dan GraphQL, sedangkan Firdaus et al. [19] mengevaluasi mekanisme mitigasi terhadap serangan pada GraphQL melalui penerapan rate limiting. Meskipun demikian, penelitian-penelitian tersebut masih memiliki beberapa keterbatasan metodologis. Pertama, sebagian besar penelitian mengevaluasi REST API dan GraphQL secara terpisah sehingga belum mampu menjelaskan perbedaan karakteristik risiko keamanan yang muncul akibat perbedaan mekanisme parsing, execution, dan schema discovery pada kedua arsitektur. Kedua, pengujian umumnya hanya dilakukan pada sistem yang rentan (vulnerable) tanpa membandingkannya dengan versi yang telah dimitigasi (fixed), sehingga efektivitas mekanisme pengamanan belum dapat divalidasi secara empiris. Ketiga, belum terdapat penelitian yang secara khusus menganalisis bagaimana fitur GraphQL introspection memengaruhi kemudahan penemuan (Ease of Discovery) kerentanan otorisasi menggunakan kerangka penilaian risiko yang terstandarisasi seperti OWASP Risk Rating Methodology. Kesenjangan metodologis tersebut menjadi landasan penelitian ini dalam melakukan analisis komparatif keamanan REST API dan GraphQL secara lebih menyeluruh.

Penelitian ini hadir untuk menjembatani kesenjangan tersebut dengan menganalisis serta membandingkan profil kerentanan REST API dan GraphQL pada tiga pilar utama OWASP API Security Top 10 2023, yakni API1 (BOLA), API3 (Mass Assignment), dan API5 (BFLA) [20]. Sebagai objek pengujian, penelitian ini mengembangkan aplikasi TixVuln merupakan sebuah platform pemesanan tiket event berbasis web yang dirancang khusus dengan kerentanan by design yang mengimplementasikan kedua arsitektur API secara bersamaan dalam satu lingkungan pengujian terkontrol. Nilai kebaruan dari riset ini terletak pada metodologi pengujian yang mengevaluasi dua kondisi sistem secara paralel: versi vulnerable yang mereplikasi celah keamanan secara autentik, dan versi fixed yang mengimplementasikan mekanisme mitigasi berupa object-level authorization, field whitelisting, dan Role-Based Access Control (RBAC). Melalui pendekatan ini, tujuan yang ingin dicapai adalah menghasilkan data empiris yang tidak hanya membuktikan eksistensi kerentanan pada kedua arsitektur, tetapi sekaligus memvalidasi efektivitas rekomendasi mitigasi yang diusulkan secara terukur. Diharapkan hasil penelitian ini dapat menjadi panduan teknis yang berbasis bukti (evidence-based) bagi para pengembang dan praktisi keamanan dalam memilih, mengimplementasikan, dan mengamankan arsitektur API yang sesuai dengan

Journal of Information System Research (JOSH)
Volume 7, No. 4, Juli 2026, pp 934−943
ISSN 2686-228X (media online)
https://ejurnal.seminar-id.com/index.php/josh/
DOI 10.47065/josh.v7i4.9874

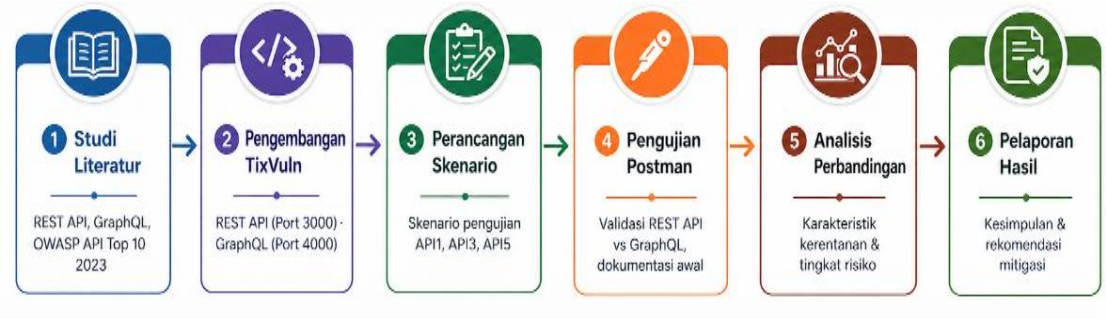
kebutuhan sistem informasi modern. arsitektur API yang sesuai dengan kebutuhan sistem informasi modern. Adapun kontribusi utama dari penelitian ini mencakup penyediaan instrumen testbed arsitektur paralel (TixVuln) yang terisolasi dari bias basis data eksternal, serta perumusan panduan mitigasi komparatif tingkat resolver untuk menambal kelemahan inheren pada arsitektur single-endpoint.

2. METODOLOGI PENELITIAN

2.1 Tahapan Penelitian

Metodologi penelitian ini mengadopsi desain eksperimental komparatif yang disusun secara sistematis melalui empat fase utama. Tahap awal (Persiapan) berfokus pada studi literatur mendalam guna memahami fundamental REST API, GraphQL, serta kerangka kerja keamanan OWASP API 2023 [21]. Tahap kedua mencakup pengembangan instrumen uji TixVuln yang dilanjutkan dengan perumusan skenario pengujian spesifik untuk setiap kategori celah keamanan.

Selanjutnya, pada tahap eksekusi, dilakukan pengujian penetrasi menggunakan Postman secara paralel untuk memvalidasi performa sistem dalam kondisi vulnerable maupun fixed. Fase Analisis Perbandingan kemudian dilaksanakan untuk mengevaluasi profil risiko teknis serta efektivitas mekanisme mitigasi yang diterapkan. Seluruh rangkaian riset ditutup dengan tahap Pelaporan Hasil yang merumuskan kesimpulan strategis dan rekomendasi penelitian, dengan alur visual yang tertera pada Gambar 1.



Gambar 1.Tahapan Penelitian

2.2 Objek Penelitian: Aplikasi TixVuln

TixVuln adalah aplikasi pemesanan tiket event berbasis web yang dikembangkan khusus untuk penelitian ini. Aplikasi ini dibangun menggunakan runtime Node.js v18 LTS sebagai backend server dan mengimplementasikan dua arsitektur API secara bersamaan dalam satu proyek, yaitu REST API yang berjalan pada port 3000 menggunakan framework Express.js v4, serta GraphQL API yang berjalan pada port 4000 menggunakan Apollo Server v4. Kedua arsitektur API berbagi penyimpanan data berbasis in-memory array JavaScript. Pemilihan in-memory array sebagai media penyimpanan dilakukan secara sadar dalam rangka mengisolasi variabel yang diuji: penelitian ini secara spesifik mengevaluasi kerentanan pada lapisan otorisasi logika API, bukan pada lapisan query database. Pendekatan ini konsisten dengan metodologi yang digunakan pada platform penelitian keamanan API terstandar seperti OWASP WebGoat dan OWASP crAPI, yang juga menggunakan basis data tersimpan di memori guna mengeleminasi variabilitas yang disebabkan oleh perbedaan implementasi database engine. Dengan demikian, setiap perbedaan risk score yang terukur dapat secara definitif diatribusikan pada perbedaan arsitektur API, bukan pada faktor eksternal lainnya. Berbeda dengan aplikasi rentan standar tunggal seperti Damn Vulnerable GraphQL Application (DVGA) yang hanya memiliki satu arsitektur, TixVuln dirancang secara khusus sebagai testbed arsitektur paralel. Hal ini memungkinkan komparasi apple-to-apple dalam satu basis data memori yang sama

Perlu ditegaskan pula bahwa kerentanan BOLA yang dikaji dalam penelitian ini beroperasi pada lapisan logika otorisasi API, di mana server gagal memvalidasi kepemilikan objek sebelum mengembalikan datasebuah kerentanan yang bersifat independen dari implementasi database yang digunakan. Berbeda dengan serangan injeksi yang mengeksploitasi kelemahan pada query database, serangan BOLA pada penelitian ini sepenuhnya beroperasi melalui manipulasi parameter pada lapisan HTTP request, sehingga penggunaan in-memory array tidak mengurangi validitas ekologis temuan penelitian ini. Dengan kata lain, apabila in-memory array digantikan dengan basis data relasional seperti MySQL atau PostgreSQL, perilaku kerentanan BOLA pada lapisan otorisasi logika API akan tetap identik karena titik eksploitasinya berada pada endpoint atau resolver, bukan pada mekanisme penyimpanan data.

Sistem autentikasi pada aplikasi ini menggunakan JSON Web Token (JWT) dengan masa berlaku selama 30 hari, sedangkan mekanisme pengamanan password menggunakan algoritma hashing bcryptjs dengan salt rounds sebesar 8. Pengujian penetrasi dilakukan secara manual berkonteks (context-aware manual testing) menggunakan Postman v10. Pendekatan manual ini dipilih karena instrumen pemindaian keamanan otomatis

Journal of Information System Research (JOSH)
Volume 7, No. 4, Juli 2026, pp 934−943
ISSN 2686-228X (media online)
https://ejurnal.seminar-id.com/index.php/josh/
DOI 10.47065/josh.v7i4.9874

(SAST/DAST) terbukti memiliki tingkat false-negative yang sangat tinggi dan kerap gagal mendeteksi cacat logika otorisasi tingkat objek (BOLA/BFLA) yang sangat bergantung pada pemahaman konteks hierarki pengguna. Implementasi setiap endpoint dan resolver dalam dua versi, yaitu vulnerable yang secara sengaja mengandung celah keamanan, serta versi fixed yang telah dilengkapi dengan mekanisme mitigasi seperti field whitelisting, Role-Based Access Control (RBAC), dan validasi kepemilikan objek (object ownership validation) [22].

2.3 Kriteria Penilaian Kerentanan

Untuk mengukur peningkatan kerentanan secara numerik dan objektif, penelitian ini menggunakan OWASP Risk Rating Methodology. Penilaian dihitung berdasarkan formula: Risk Score = Likelihood (Probabilitas) x Impact (Dampak). Nilai Likelihood dan Impact diukur pada skala 1 hingga 9 berdasarkan kemudahan eksploitasi (ease of exploit) dan kemudahan penemuan (ease of discovery)[23], [24]. Hasil perkalian akan menentukan tingkat keparahan dengan rentang: Rendah (1-15), Sedang (16-30), Tinggi (31-60), dan Kritis (61-81).

Penilaian terhadap tingkat keamanan API dalam penelitian ini diklasifikasikan ke dalam dua parameter utama. Status Vulnerable ditetapkan apabila upaya eksploitasi dinyatakan berhasil, yang secara teknis divalidasi melalui penerimaan respons HTTP 200 OK atau terpaparnya data sensitif pada response body akibat penembusan kontrol akses terhadap fungsi maupun data ilegal. Di sisi lain, status Protected merepresentasikan keberhasilan sistem dalam memitigasi serangan. Indikator perlindungan ini ditandai dengan kemunculan kode status HTTP 403 Forbidden, 404 Not Found, atau pesan kesalahan (error) spesifik pada resolver GraphQL. Selain itu, efektivitas proteksi juga dibuktikan melalui mekanisme whitelist yang secara otomatis mengabaikan atau menolak setiap upaya modifikasi pada atribut (field) sensitif yang dikirimkan oleh penyerang.

2.4 Akun dan Data Pengujian

Pengujian dilakukan menggunakan tiga akun yang masing-masing memiliki peran berbeda dalam skenario serangan. pengujian dilakukan dalam dua tahap: tahap eksploitasi (pada endpoint rentan) dan tahap verifikasi (pada endpoint yang telah diperbaiki). Rincian detail akun pengujian tersebut ditunjukkan pada Tabel 1.

Tabel 1. Akun Pengujian TixVuln

Username	Password	Role	ID	Peran dalam Pengujian
admin	admin123	Admin	1	Target data
alice	alice123	User	2	Attacker (penyerang)
bob	bob123	User	3	Pengguna biasa

2.5 Skenario Pengujian

Setiap kategori OWASP diuji menggunakan dua kondisi utama: (1) skenario serangan pada titik akhir (endpoint) yang rentan (vulnerable) untuk membuktikan keberadaan celah, dan (2) skenario verifikasi pada versi yang telah diperbaiki (fixed) sebagai pembanding efektivitas mitigasi. Penambahan versi fixed pada setiap kategori (API1, API3, dan API5) bertujuan untuk memvalidasi bahwa rekomendasi mitigasi yang diusulkan, seperti object-level authorization, field whitelisting, dan Role-Based Access Control (RBAC), benar-benar mampu menutup celah keamanan pada kedua arsitektur API tersebut. Rincian deskripsi skenario pengujian untuk setiap kategori diuraikan secara lengkap pada Tabel 2.

Tabel 2. Skenario Pengujian API1,API3,dan API5

Kategori	Versi	Deskripsi Skenario Pengujian
API 1: BOLA	Normal	Mengakses data profil milik sendiri melalui ID pengguna yang sah.
	Vulnerable	Mengakses data profil admin dengan memanipulasi parameter ID pada URL atau query.
	Fixed	Verifikasi penolakan akses saat mencoba mengakses ID milik pengguna lain.
API3 : Mass Assignment	Normal	Melakukan pembaruan profil standar (seperti nama atau alamat)
	Vulnerable	Mencoba mengubah hak akses dengan menyisipkan atribut role: admin pada body request.
	Vulnerable	Mencoba menambah saldo secara ilegal dengan menyisipkan atribut balance pada body request
	Fixed	Verifikasi bahwa atribut sensitif diabaikan oleh sistem melalui mekanisme field whitelisting
API5: BFLA	Normal	Mengakses fungsi standar yang diizinkan untuk level pengguna biasa.
	Vulnerable	Mencoba memanggil fungsi administratif untuk melihat daftar seluruh pengguna sistem.
	Vulnerable	Mencoba memanggil fungsi administratif untuk menghapus data pengguna lain

Journal of Information System Research (JOSH)
Volume 7, No. 4, Juli 2026, pp 934−943
ISSN 2686-228X (media online)
https://ejurnal.seminar-id.com/index.php/josh/
DOI 10.47065/josh.v7i4.9874

Kategori	Versi	Deskripsi Skenario Pengujian
	Fixed	Verifikasi blokir akses pada fungsi administratif melalui pengecekan Role-Based Access Control (RBAC)

3. HASIL DAN PEMBAHASAN

3.1 Persiapan Lingkungan dan Autentikasi Pengguna

Tahap krusial pertama dalam eksploitasi otorisasi API adalah proses autentikasi untuk membedakan hak akses antar pengguna. Pengujian ini menggunakan tiga identitas utama: Admin (ID: 1) sebagai target pencurian data, Alice (ID: 2) sebagai penyerang dengan hak akses pengguna biasa, dan Bob (ID: 3) sebagai entitas sekunder. Proses pengujian dimulai dengan Alice mengirimkan kredensial melalui request POST ke /api/auth/login (untuk REST) dan mutation login (untuk GraphQL). Kedua arsitektur merespons dengan menerbitkan JSON Web Token (JWT) yang valid. Diperolehnya token ini mensimulasikan skenario ancaman internal (insider threat), di mana penyerang sudah masuk ke dalam sistem secara sah, sehingga evaluasi terhadap kontrol otorisasi pada tahap selanjutnya menjadi sangat relevan dan valid.

3.2 Hasil Pengujian API1 Broken Object Level Authorization (BOLA)

Pengujian API1 bertujuan memverifikasi apakah sistem memvalidasi hak akses pengguna terhadap objek data milik pengguna lain. Skenario serangan dilakukan dengan pengguna alice (ID=2) mencoba mengakses data profil admin (ID=1).

Pada REST API, serangan dilakukan dengan memodifikasi nilai ID pada URL dari /api/users/2 menjadi /api/users/1 menggunakan method GET. Seperti ditunjukkan pada Gambar 3, server merespons dengan HTTP 200 OK dan mengembalikan seluruh data profil admin termasuk informasi sensitif berupa role, balance, dan email. Ini mengonfirmasi adanya kerentanan BOLA pada REST API. Sementara itu, pada pengujian terhadap endpoint yang telah diperbaiki (/api/users/1/secure), server memberikan respons HTTP 404 Not Found. Hasil ini mengonfirmasi bahwa mekanisme proteksi berhasil dijalankan dengan cara menutup akses atau menyembunyikan rute tersebut dari pengguna yang tidak memiliki wewenang, sehingga penyerang tidak mendapatkan informasi apa pun mengenai keberadaan sumber daya tersebut.

Pada GraphQL, serangan ekuivalen dilakukan dengan mengubah argumen id dari 2 menjadi 1 pada query: query { user(id: 1) { ... } }dengan HTTP 200 OK beserta data admin lengkap. Pengujian resolver fixed (userSecure) mengembalikan pesan error "Forbidden: Anda tidak memiliki akses ke data ini". Perbedaan signifikan yang ditemukan adalah pada aspek deteksi: serangan REST API mudah teridentifikasi melalui log URL karena perubahan ID terlihat jelas, sementara pada GraphQL seluruh request baik normal maupun serangan dikirim ke endpoint yang sama (/graphql) sehingga lebih sulit dibedakan berdasarkan URL. Rekapitulasi hasil pengujian serangan BOLA menggunakan Postman beserta pembuktian efektivitas mitigasi Object Ownership Check pada kedua arsitektur disajikan pada Tabel 3.

Tabel 3. Hasil Pengujian API1 (BOLA)

Skenario Pengujian	Arsitektur	Endpoint / Payload	HTTP	Status	Severity (Skor)
Baseline (Akses data sendiri)	REST API	GET /api/users/2	200	Normal	-
Eksploitasi BOLA (Akses Admin)	REST API	GET /api/users/1	200	Vulnerable	Tinggi (42)
Pengujian Versi Mitigasi	REST API	GET /api/users/1/secure	404	Protected	Aman (0)
Baseline (Akses data sendiri)	GraphQL	query { user(id:2) }	200	Normal	-
Eksploitasi BOLA (Akses Admin)	GraphQL	query { user(id:1) }	200	Vulnerable	Tinggi (42)
Pengujian Versi Mitigasi	GraphQL	query { userSecure(id:1) }	Error	Protected	Aman (0)

Meskipun dampak dari kebocoran ini sama parahnya, REST API dinilai lebih transparan karena modifikasi ID terlihat jelas pada log sistem pengawasan lalu lintas jaringan (traffic log). Pada versi mitigasi (fixed), penambahan logika kode if (req.user.id !== targetId) return res.status(404) secara drastis menggagalkan semua upaya enumerasi ID tersebut.

3.3 Hasil Pengujian API3 Broken Object Property Level Authorization (Mass Assignment)

Pengujian kategori API3 dilakukan dengan skenario di mana pengguna alice mencoba memanipulasi properti sensitif berupa role dan balance melalui permintaan pembaruan data pengguna. Setiap skenario eksploitasi diikuti dengan pengujian pada versi fixed sebagai pembanding efektivitas mitigasi. Ringkasan hasil eksploitasi kerentanan

Journal of Information System Research (JOSH)
Volume 7, No. 4, Juli 2026, pp 934−943
ISSN 2686-228X (media online)
https://ejurnal.seminar-id.com/index.php/josh/
DOI 10.47065/josh.v7i4.9874

Mass Assignment dan pengujian versi mitigasi menggunakan mekanisme field whitelisting dapat dilihat secara detail pada Tabel 4.

Tabel 4. Hasil Pengujian API3 (Mass Assignment)

Skenario Pengujian	Arsitektur	Endpoint / Payload	HTTP	Status	Severity (Skor)
Baseline (Update profil normal)	REST API	PUT /api/users/2	200	Normal	-
Injeksi JSON Body (role)	REST API	{"role": "admin"}	200	Vulnerable	Sedang (24)
Injeksi JSON Body (balance)	REST API	{"balance": 99999}	200	Vulnerable	Sedang (24)
Pengujian Versi Mitigasi	REST API	PUT /api/users/2/secure	200	Protected	Aman (0)
Baseline (Update profil normal)	GraphQL	mutation { updateUser }	200	Normal	-
Injeksi Argumen (role)	GraphQL	role: "admin"	200	Vulnerable	Tinggi (48)
Injeksi Argumen (balance)	GraphQL	balance: 99999	200	Vulnerable	Tinggi (48)
Pengujian Versi Mitigasi	GraphQL	mutation { updateUserSecure }	Error	Protected	Aman (0)

Pada arsitektur REST API, serangan dilakukan melalui rute PUT /api/users/2 dengan mengirimkan request body berupa {"role": "admin", "balance": 99999999}. Hasil pengujian menunjukkan bahwa server menerima dan memproses seluruh field tersebut tanpa adanya validasi otoritas properti, yang mengakibatkan tingkatan akses alice berubah menjadi "admin" dan saldo akun bertambah secara ilegal sesuai nilai yang diinjeksikan. Sebaliknya, pengujian pada versi fixed (PUT /api/users/2/secure) membuktikan bahwa meskipun penyerang mengirimkan field role dan balance, server hanya memproses atribut yang terdaftar dalam whitelist (seperti phone dan address) dan secara otomatis mengabaikan atribut sensitif lainnya. Hal ini mengonfirmasi efektivitas mekanisme field whitelisting dalam menjaga integritas properti objek.

Eksploitasi pada arsitektur GraphQL dilakukan melalui mutation updateUser dengan menyisipkan parameter sensitif berupa role dan balance. Serangan Mass Assignment ini berhasil memanipulasi data pengguna secara ilegal. Namun, pengujian pada versi mitigasi (updateUserSecure) mengungkapkan keunggulan arsitektural GraphQL dibandingkan REST API. Melalui sistem Strongly Typed Schema, GraphQL secara otomatis menolak permintaan yang mengandung argumen di luar definisi skema bahkan sebelum permintaan tersebut diproses oleh resolver. Karakteristik ini menciptakan mekanisme keamanan berlapis, di mana pembatasan pada level skema (schema-level restriction) berfungsi sebagai baris pertahanan pertama, diikuti oleh validasi logika pada level resolver sebagai baris pertahanan kedua.

Disparitas kerentanan yang signifikan antara REST dan GraphQL pada kategori ini terletak pada tingkat kemudahan penemuan (ease of discovery) parameter sensitif, yang selanjutnya akan dikuantifikasi pada sub-bab penilaian risiko numerik.

3.4 Hasil Pengujian API5 Broken Function Level Authorization (BFLA)

Pengujian kategori API5 dilakukan dengan skenario di mana pengguna alice mencoba mengakses dan mengeksekusi fungsi administratif yang seharusnya terbatas bagi peran admin saja. Fokus pengujian ini mencakup fungsi penarikan data seluruh pengguna dan penghapusan akun. Hasil pengujian penetrasi pada kategori BFLA serta pembuktian ketangguhan mitigasi berbasis RBAC dirangkum pada Tabel 5.

Tabel 5. Hasil Pengujian API5 (Broken Function Level Authorization).

Skenario Pengujian	Arsitektur	Endpoint / Payload	HTTP	Status	Severity (Skor)
Baseline (Akses user biasa)	REST API	GET /api/users/	200	Normal	-
Akses Fungsi Lihat Semua User	REST API	GET /api/admin/users	200	Vulnerable	Tinggi (40)
Akses Fungsi Admin (Hapus)	REST API	DELETE /api/users/3	200	Vulnerable	Tinggi (40)
Pengujian Versi Mitigasi	REST API	DELETE /api/users/3/secure	403	Protected	Aman (0)
Baseline (Akses user biasa)	GraphQL	query { users }	200	Normal	-
Akses Fungsi Lihat Semua User	GraphQL	query { allUsers }	200	Vulnerable	Kritis (72)
Akses Fungsi Admin (Hapus)	GraphQL	mutation	200	Vulnerable	Kritis (72)

Journal of Information System Research (JOSH)
Volume 7, No. 4, Juli 2026, pp 934−943
ISSN 2686-228X (media online)
https://ejurnal.seminar-id.com/index.php/josh/
DOI 10.47065/josh.v7i4.9874

Skenario Pengujian	Arsitektur	Endpoint / Payload	HTTP	Status	Severity (Skor)
Pengujian Versi Mitigasi	GraphQL	{ deleteUser(id:3) } Mutation { deleteUserSecure }	Error	Protected	Aman (0)

Pada arsitektur REST API, alice berhasil mengakses rute GET /api/admin/users yang memaparkan data sensitif seluruh pengguna dan menjalankan perintah DELETE /api/users/3 untuk menghapus akun pengguna lain tanpa adanya validasi role di tingkat server. Temuan ini mengonfirmasi kerentanan BFLA yang memiliki skor risiko Tinggi (40). Pengujian pada versi fixed (GET /api/admin/users/secure) membuktikan efektivitas mitigasi dengan mengembalikan kode status HTTP 403 Forbidden, di mana server kini secara ketat melakukan validasi req.user.role === "admin" sebelum memproses permintaan.

Eksploitasi pada arsitektur GraphQL menunjukkan bahwa operasi administratif seperti query allUsers dan mutation deleteUser juga dapat dieksekusi oleh akun alice pada versi vulnerable. Namun, secara komparatif, GraphQL menghadirkan profil risiko yang jauh lebih signifikan dibandingkan REST API. Keberadaan fitur introspection memfasilitasi penyerang untuk memetakan seluruh struktur skema administratif melalui satu endpoint tunggal, sehingga identifikasi fungsi-fungsi administratif yang tersembunyi menjadi jauh lebih mudah dan komprehensif. Hal inilah yang memicu lonjakan skor risiko sebesar 80% hingga mencapai level Kritis (72). Pengujian pada versi mitigasi (allUsersSecure dan deleteUserSecure) terbukti efektif membendung serangan dengan mengembalikan pesan error "Forbidden: Akses hanya untuk Admin" sebagai hasil dari implementasi Role-Based Access Control (RBAC) pada level resolver.

3.5 Kuantifikasi Peningkatan Risiko (Analisis Numerik)

Untuk menjawab pertanyaan seberapa besar peningkatan kerentanan yang disebabkan oleh karakteristik arsitektur GraphQL dibandingkan REST API, penelitian ini mengkalkulasi skoring ancaman menggunakan parameter OWASP Risk Rating Methodology. Faktor Likelihood (Probabilitas) dipengaruhi kuat oleh Ease of discovery (seberapa mudah celah ditemukan). Faktor Impact (Dampak) dinilai dari kerugian bisnis atau kerusakan data jika celah berhasil dieksploitasi. Penilaian risiko dihitung menggunakan formula :

$$\text{Risk Score} = \text{Likelihood} \times \text{Impact} \quad (1)$$

Tabel 6 menyajikan kalkulasi numerik dari skor ancaman sebelum menggunakan perlindungan (versi vulnerable) untuk membandingkan lonjakan risiko antara model desain REST dan pola operasional GraphQL. Guna memastikan objektivitas dan mengeleminasi potensi confirmation bias dalam penentuan nilai pada Tabel 6, seluruh nilai risk score ditetapkan secara a priori berdasarkan definisi operasional OWASP Testing Guide sebelum sesi pengujian dilaksanakan, dan tidak dimodifikasi setelah hasil pengujian diperoleh. Penetapan nilai dilakukan berdasarkan karakteristik teknis yang dapat diverifikasi secara objektif dan deterministik: nilai 9 untuk Ease of Discovery GraphQL pada API3 dan API5 didasarkan pada fakta bahwa query introspection mengembalikan seluruh struktur skema secara lengkap dan konsisten pada setiap eksekusi tanpa variabilitas, bukan berdasarkan estimasi subjektif peneliti. Nilai 4 untuk Ease of Discovery REST API pada API3 ditetapkan karena penyerang harus menebak parameter sensitif secara manual tanpa panduan skema, sedangkan nilai 5 pada API5 ditetapkan karena memerlukan proses fuzzing aktif terhadap pola URL administratif. Perbedaan nilai inilah yang secara langsung mencerminkan perbedaan teknis inheren antara kedua arsitektur, bukan konstruksi yang dipaksakan untuk mendukung hipotesis tertentu. Untuk mencapai komparasi keamanan yang adil dan merepresentasikan kondisi industri nyata, arsitektur GraphQL perlu dievaluasi dengan asumsi standar production-ready, di mana fitur introspection dinonaktifkan secara total. Dalam kondisi introspection off, profil Ease of Discovery pada GraphQL akan menurun drastis dan setara dengan REST API, karena penyerang dipaksa melakukan eksploitasi buta (blind fuzzing). Sebaliknya, apabila arsitektur REST API mengalami miskonfigurasi dengan mengekspos dokumentasi skema interaktif (seperti antarmuka Swagger UI atau OpenAPI) secara publik di lingkungan produksi, REST API juga akan mengalami lonjakan risiko Ease of Discovery yang sama fatalnya dengan GraphQL yang mengekspos introspection. Oleh karena itu, disparitas skor pada pengujian ini murni menyoroti bahaya laten dari eksposur skema pada arsitektur single-endpoint, bukan merepresentasikan kelemahan absolut dari bahasa GraphQL itu sendiri.

Tabel 6. Perbandingan Skor Risiko Numerik REST API dan GraphQL

Kategori Kerentanan OWASP				
Arsitektur	**Likelihood (1-9)**	**Impact (1-9)**	**Risk Score Numerik**	**Kategori Resiko**
		API1: BOLA		
REST API	6 (Sedang)	7 (Tinggi)	42	Tinggi
GraphQL	6 (Sedang)	7 (Tinggi)	42	Tinggi
		API3: Mass Assigment		
REST API	4 (Rendah - Menebak)	6 (Tinggi)	24	Sedang

Journal of Information System Research (JOSH)
Volume 7, No. 4, Juli 2026, pp 934−943
ISSN 2686-228X (media online)
https://ejurnal.seminar-id.com/index.php/josh/
DOI 10.47065/josh.v7i4.9874

Kategori Kerentanan OWASP				
Arsitektur	**Likelihood (1-9)**	**Impact (1-9)**	**Risk Score Numerik**	**Kategori Resiko**
GraphQL	8 (Tinggi - Terbaca)	6 (Tinggi) API: BFLA	48	Tinggi
REST API	5 (Sedang - Fuzzing)	8 (Kritis)	40	Tinggi
GraphQL	9 (Sangat Tinggi)	8 (Kritis)	72	Kritis

Analisis peningkatan risiko berdasarkan Tabel 6 menghasilkan temuan esensial: Pada kerentanan API1 (BOLA), tingkat risiko tetap stabil pada kategori Tinggi untuk kedua arsitektur. Namun, pada API3 (Mass Assignment) dan API5 (BFLA), penggunaan GraphQL memicu lompatan kategori risiko (risk tier leap) yang sangat drastis, yakni dari kategori Sedang menjadi Tinggi pada API3, dan dari kategori Tinggi melompat hingga ambang batas Kritis pada API5. Lompatan signifikansi metrik kemudahan penemuan ini dipicu secara absolut oleh eksposur skema GraphQL yang membuat parameter tersembunyi menjadi terpampang jelas, memfasilitasi serangan fungsi presisi (precision function attack) tanpa perlu melakukan blind fuzzing.

3.6 Perbandingan Karakteristik Kerentanan REST API vs GraphQL

Secara arsitektural, perbedaan pola komunikasi antara REST API dan GraphQL melahirkan pendekatan yang berbeda pula dalam hal identifikasi dan eksploitasi celah keamanan. Tabel 7 berikut merangkum perbandingan karakteristik teknis dari kedua arsitektur tersebut berdasarkan lima aspek fundamental keamanan API, yang mencakup struktur endpoint, visibilitas skema, bentuk vektor serangan, luas permukaan serangan (attack surface), hingga strategi mitigasi yang paling optimal.

Tabel 7 . Perbandingan Karakteristik Kerentanan REST API dan GraphQL

No	Aspek Perbandingan	REST API	GraphQL
1	Struktur Endpoint	Tersebar pada banyak rute URL (Multiple Endpoints).	Terpusat pada satu titik akses (Single Endpoint, umumnya /graphql).
2	Visibilitas Skema (Discovery)	Tertutup. Penyerang harus melakukan fuzzing atau menebak rute secara manual.	Terbuka luas. Skema dapat dipetakan secara instan menggunakan fitur Introspection.
3	Vektor Serangan (Payload)	Manipulasi parameter pada URL (Path/Query) dan injeksi data pada JSON Body.	Manipulasi argumen Query dan Mutation di dalam request body.
4	Permukaan Serangan (Attack surface)	Terletak pada titik-titik URL administratif yang tidak terproteksi.	Terletak pada logika internal resolver yang memproses operasi basis data.
5	Strategi Mitigasi Optimal	Validasi input pada middleware routing dan proteksi rute berbasis HTTP (RBAC).	Penonaktifan Introspection, validasi di tingkat resolver, dan field whitelisting.

Perbedaan paling signifikan ditemukan pada dampak fitur introspection GraphQL yang menjadi vektor risiko tambahan pada kategori API3 dan API5. Temuan ini sejalan dengan penelitian Napisa et al. yang mengidentifikasi information disclosure vulnerability sebagai ancaman khas pada ekosistem GraphQL [14].

Karakteristik ini pula yang secara matematis memicu lonjakan skor risiko hingga 100% pada pengujian Mass Assignment di penelitian ini. Adapun kerentanan pada kategori BOLA (API1) ditemukan konsisten pada kedua arsitektur, selaras dengan hasil kajian Santos Filho et al. yang menyatakan bahwa broken object-level authorization merupakan celah persisten pada REST API yang sulit dideteksi secara otomatis [15]. Riset ini memperluas temuan tersebut dengan menunjukkan bahwa kerentanan serupa hadir pada GraphQL dengan karakteristik vektor serangan yang berbeda namun memiliki dampak risiko yang setara.

Kontribusi utama penelitian ini adalah pembuktian empiris mengenai efektivitas mitigasi melalui pengujian versi fixed. Berbeda dengan penelitian Prasetyo dan Kriestanto yang membandingkan GraphQL dan REST API hanya dari perspektif performa tanpa menyertakan analisis keamanan [5], penelitian ini memfokuskan komparasi pada dimensi kerentanan otorisasi. Selain itu, berbeda pula dengan Firdaus et al. yang mengkaji proteksi GraphQL terbatas pada serangan DDoS melalui rate limiting [19], riset ini mencakup tiga kategori kerentanan otorisasi yang lebih fundamental. Hasil pengujian menunjukkan bahwa implementasi object-level authorization check, field whitelisting, dan Role-Based Access Control (RBAC) yang tepat berhasil mencegah seluruh serangan pada kedua arsitektur. Hal ini memperkuat rekomendasi Mazidi et al. bahwa Mass assignment vulnerability pada API memerlukan mekanisme validasi eksplisit baik di level skema maupun logika aplikasi [16].

Temuan terkait tingginya risiko fitur introspection pada GraphQL dalam konteks API5 juga didukung oleh kajian Baihan yang menekankan pentingnya implementasi RBAC yang ketat pada sistem berbasis GraphQL, mengingat sifat single-endpoint-nya yang memudahkan pemetaan seluruh fungsi oleh pihak tidak berwenang [7]. Sementara itu, dari sisi perbandingan attack surface, Zhao menegaskan bahwa pengelolaan permukaan serangan merupakan komponen kritis dalam strategi keamanan API modern [9]. Penelitian ini membuktikan secara empiris

Journal of Information System Research (JOSH)
Volume 7, No. 4, Juli 2026, pp 934−943
ISSN 2686-228X (media online)
https://ejurnal.seminar-id.com/index.php/josh/
DOI 10.47065/josh.v7i4.9874

bahwa single-endpoint GraphQL dengan introspection aktif menciptakan attack surface yang lebih luas dan skor risiko yang lebih tinggi (Kritis) dibandingkan struktur multiple-endpoint pada REST API.

3.7 Pembahasan

Temuan empiris dalam penelitian ini memperluas dan menyempurnakan hasil riset-riset terdahulu mengenai keamanan API. Berbeda dengan kajian Napisa et al. [14] yang sebatas mengidentifikasi ancaman information disclosure pada fitur introspection GraphQL, penelitian ini memberikan kontribusi kebaruan dengan mengkuantifikasi dampak lanjutan dari paparan informasi tersebut terhadap eskalasi kerentanan otorisasi tingkat objek dan fungsi (API3 dan API5). Selanjutnya, bila dibandingkan dengan penelitian Santos Filho et al. [15] dan Mazidi et al. [16] yang secara eksklusif berfokus merumuskan pendeteksian kerentanan BOLA dan Mass Assignment pada REST API konvensional, riset ini memperluas batasan ilmu tersebut dengan membuktikan bahwa kerentanan identik juga mengancam ekosistem GraphQL, namun dengan metrik kemudahan penemuan (Ease of Discovery) yang melonjak drastis hingga level Kritis. Selain itu, temuan ini secara langsung membantah asumsi teknis yang kerap muncul dalam literatur bahwa efisiensi performa GraphQL yang dikaji oleh Pratama et al. [17] dan Sadaf et al. [18] berbanding lurus dengan keamanan bawaan arsitekturnya. Sebaliknya, penelitian ini secara kuat membuktikan peringatan Baihan [7]: fleksibilitas arsitektur single-endpoint GraphQL menuntut penerapan mekanisme validasi kontrol akses (RBAC dan field whitelisting) yang jauh lebih terdesentralisasi ke tingkat resolver, tidak bisa lagi hanya bergantung pada pelindung routing middleware jaringan konvensional.

4. KESIMPULAN

Berdasarkan hasil analisis komparatif menggunakan kerangka kerja OWASP API Security Top 10 2023, penelitian ini mengonfirmasi bahwa arsitektur REST API dan GraphQL secara fundamental memiliki kerentanan dasar yang setara secara spesifik pada lapisan logika otorisasi. Namun, pengabaian praktik keamanan standar berupa fitur introspection aktif pada GraphQL terbukti memicu eskalasi kategori risiko secara drastis melompat hingga menyentuh level Kritis pada API3 dan API5 akibat terekspos nya permukaan serangan secara absolut. Sebagai sintesis dari temuan empiris tersebut, penelitian ini merumuskan kerangka kerja keamanan Resolver-Centric Defense-in-Depth khusus untuk ekosistem single-endpoint. Kerangka ini menggarisbawahi bahwa keamanan GraphQL tidak dapat lagi mengandalkan perlindungan tingkat routing jaringan seperti pada REST API, melainkan menuntut integrasi object-level authorization, field whitelisting, dan Role-Based Access Control (RBAC) yang ditanamkan secara ketat pada setiap lapisan resolver fungsional. Bertolak dari sintesis ini, implikasi manajerial bagi organisasi yang merencanakan migrasi arsitektur dari REST ke GraphQL adalah ketidakmampuan strategi "lift-and-shift" dalam mentransfer kebijakan keamanan lama. Manajemen tingkat eksekutif harus mengorkestrasi pergeseran paradigma menuju security-by-design, yang mencakup audit skema otorisasi internal serta penegakan kebijakan operasional absolut untuk menonaktifkan fitur introspection di lingkungan produksi. Sebagai rekomendasi pengembangan akademis di masa depan, penelitian selanjutnya perlu difokuskan pada perancangan algoritma deteksi anomali query berbasis analisis perilaku (behavioral analysis) untuk mengidentifikasi ancaman precision function attack pada lalu lintas data sentralisasi yang kompleks. Sebagai kontribusi keilmuan dan praktis, penelitian ini menyediakan landasan empiris baru bagi praktisi keamanan bahwa adopsi arsitektur API modern tidak boleh hanya didorong oleh motif efisiensi performa, melainkan wajib diimbangi dengan restrukturisasi strategi mitigasi ancaman logika bisnis yang disesuaikan dengan mekanisme spesifik parsing arsitektur tersebut.

REFERENCES

[1] F. X. Senduk, X. B. N. Najoan, and S. R. U. A. Sompie, “Pengembangan Arsitektur Microservices dengan RESTful API Gateway menggunakan Backend-for-frontend Pattern pada Portal Akademik Perguruan Tinggi,” J. Tek. Inform., vol. 18, no. 1, pp. 315–324, Mar. 2023, doi: 10.35793/jti.v18i1.50402.

[2] Mohammed Mudassir and Mohammed Mushtaq, “The role of APIs in modern software development,” World J. Adv. Eng. Technol. Sci., vol. 13, no. 1, pp. 1045–1047, Oct. 2024, doi: 10.30574/wjaets.2024.13.1.0515.

[3] S. Kanthed, “Rest vs. GraphQL: Comparative Analysis of API Design Approaches,” Int. J. Multidiscip. Res. Growth Eval., vol. 4, no. 1, pp. 984–991, 2023, doi: 10.54660/.IJMRGE.2023.4.1.984-991.

[4] N. Vohra and I. B. Kerthyayana Manuaba, “Implementation of REST API vs GraphQL in Microservice Architecture,” in 2022 International Conference on Information Management and Technology (ICIMTech), IEEE, Aug. 2022, pp. 45–50. doi: 10.1109/ICIMTech55957.2022.9915098.

[5] A. Prasetyo and D. Kriestanto, “Analisis Perbandingan GraphQL dan REST API pada Aplikasi Menu Restoran dengan Node.js,” JIKO (Jurnal Inform. dan Komputer), vol. 9, no. 2, p. 274, Jun. 2025, doi: 10.26798/jiko.v9i2.1521.

[6] Nagaraju Thallapally, “Enhancing Data Query Flexibility with GraphQL: Implementation and Best Practices,” J. Comput. Sci. Technol. Stud., vol. 6, no. 2, pp. 176–182, Jun. 2024, doi: 10.32996/jcsts.2024.6.2.20.

[7] M. S. Baihan, “Role-based Access Control Solution for GraphQL-based Fast Healthcare Interoperability Resources Health Application Programming Interface,” in 2022 IEEE International Conference on E-health Networking, Application & Services (HealthCom), IEEE, Oct. 2022, pp. 1–6. doi: 10.1109/HealthCom54947.2022.9982782.

[8] M. Muntahanah, Y. Darmi, and K. Pinandita, “Implementasi Perbandingan Metode Graphql Dan Rest Api Pada Teknologi Nodejs,” INTECOMS J. Inf. Technol. Comput. Sci., vol. 7, no. 1, pp. 25–34, Jan. 2024, doi: 10.31539/intecoms.v7i1.8656.

Journal of Information System Research (JOSH)
Volume 7, No. 4, Juli 2026, pp 934−943
ISSN 2686-228X (media online)
https://ejurnal.seminar-id.com/index.php/josh/
DOI 10.47065/josh.v7i4.9874

[9] C. Zhao, "API Common Security Threats and Security Protection Strategies," Front. Comput. Intell. Syst., vol. 10, no. 2, pp. 29–33, Nov. 2024, doi: 10.54097/k5djs164.
[10] D. Kamble, "Exploring the Threat Landscape of API Attacks," Int. J. Sci. Res. Eng. Manag., vol. 09, no. 06, pp. 1–9, Jun. 2025, doi: 10.55041/IJSREM50004.
[11] Ishva Jitendrakumar Kanani, "Securing APIs in the modern threat landscape: Best practices and challenges," World J. Adv. Res. Rev., vol. 13, no. 3, pp. 654–657, Mar. 2022, doi: 10.30574/wjarr.2022.13.3.0239.
[12] Y. Ishida, M. Hanada, A. Waseda, and M. W. Kim, "Automated Vulnerability Assessment Approach for Web API that Considers Requests and Responses," in 2024 26th International Conference on Advanced Communications Technology (ICACT), IEEE, Feb. 2024, pp. 1521–1533. doi: 10.23919/ICACT60172.2024.10471939.
[13] Suresh Vethachalam, "How hackers exploit poorly built APIs – A developer's guide to API Hardening," World J. Adv. Eng. Technol. Sci., vol. 10, no. 2, pp. 426–440, Dec. 2023, doi: 10.30574/wjaets.2023.10.2.0290.
[14] R. Napisa, A. Widjajarto, and U. Y. K. S. Hediyanto, "Desain Attack Tree Berdasar Metrik Time Pada Eksploitasi GraphQL Dengan Information Disclosure Vulnerability," J. Teknol. Dan Sist. Inf. Bisnis, vol. 7, no. 2, pp. 310–319, Apr. 2025, doi: 10.47233/jteksis.v7i2.1627.
[15] A. Santos Filho, R. J. Rodríguez, and E. L. Feitosa, "Automated broken object-level authorization attack detection in REST APIs through OpenAPI to colored petri nets transformation," Int. J. Inf. Secur., vol. 24, no. 2, p. 83, Apr. 2025, doi: 10.1007/s10207-024-00970-5.
[16] A. Mazidi, D. Corradini, and M. Ghafari, "Mining REST APIs for Potential Mass Assignment Vulnerabilities," in Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering, New York, NY, USA: ACM, Jun. 2024, pp. 369–374. doi: 10.1145/3661167.3661204.
[17] I. G. A. E. Pratama, I. P. Satwika, and I. N. Y. Anggara Wijaya, "Analisis Perbandingan Performa API Metode REST dan GraphQL dengan PHP dan Go," J. Teknol. Inf. dan Komput., vol. 8, no. 4, Oct. 2022, doi: 10.36002/jutik.v8i4.2088.
[18] K. Sadaf, S. Khan, D. Chourasia, N. Rai, and M. Jamal, "Comparison of Rest vs Graphql: Performance, Authentication and Scalability," in Proceedings of the 1st International Conference on Research and Development in Information, Communication, and Computing Technologies, SCITEPRESS - Science and Technology Publications, 2025, pp. 369–377. doi: 10.5220/0013930100004919.
[19] Diash Firdaus, I. Sumardi, and G. Nugraha, "Peningkatan Keamanan Server GraphQL Terhadap Serangan DDOS Dengan Tipe Batch Attack Menggunakan Metode Rate Limiting," Cyber Secur. dan Forensik Digit., vol. 7, no. 2, pp. 62–68, Dec. 2024, doi: 10.14421/csecurity.2024.7.2.4718.
[20] M. Idris, I. Syarif, and I. Winarno, "Development of Vulnerable Web Application Based on OWASP API Security Risks," in 2021 International Electronics Symposium (IES), IEEE, Sep. 2021, pp. 190–194. doi: 10.1109/IES53407.2021.9593934.
[21] P. Bhosale, "Implementing Secure and Scalable APIs in Java Spring Boot: RESTful vs. GraphQL Services," INTERANTIONAL J. Sci. Res. Eng. Manag., vol. 09, no. 01, pp. 1–7, Jan. 2025, doi: 10.55041/IJSREM11316.
[22] Y. Shcheblanin, B. Sydorenko, and I. Mykhalchuk, "Security of REST API: Threats and Protection Methods," Inf. Syst. Technol. Secur., no. 2 (8), pp. 60–65, 2024, doi: 10.17721/ISTS.2024.8.60-65.
[23] M. Saifulhakim and A. Nurul Fajar, "Security Assessment for Digital Wallet Payment Partner Applications using the OWASP Method: A Case Study in Indonesia," J. Syst. Manag. Sci., vol. 14, no. 3, Feb. 2024, doi: 10.33168/JSMS.2024.0319.
[24] M. Idris, I. Syarif, and I. Winarno, "Web Application Security Education Platform Based on OWASP API Security Project," Emit. Int. J. Eng. Technol., pp. 246–261, Dec. 2022, doi: 10.24003/emitter.v10i2.705.