CyberSecurity dan Forensik Digital e-ISSN: 2615-8442
Vol. 9, No. 1, Mei 2026, hlm. 1-9

Optimasi Sistem Keamanan REST API Menggunakan Zero Trust Architecture Dan Keycloak Tools Dalam Infrastruktur Berbasis Microservices

Denny Afrizal[1], Muhammad Taufiq Nuruzzaman[2], Bambang Sugiantoro[3], Agung Fatwanto[4]

[1] Magister Informatika, Fakultas Sains dan Teknologi, UIN Sunan Kalijaga, Yogyakarta, Indonesia
[2,3,4] Teknik Informatika, Fakultas Sains dan Teknologi, UIN Sunan Kalijaga, Yogyakarta, Indonesia
Email: [1]denny.afrizal713@gmail.com, [2]m.taufiq@uin-suka.ac.id, [3]bambang.sugiantoro@uin-suka.ac.id, [4]agung.fatwanto@uin-suka.ac.id

Abstrak

Keamanan layanan *REST API* pada arsitektur *microservices* menjadi isu penting seiring meningkatnya kompleksitas sistem digital dan perluasan permukaan serangan. Penelitian ini bertujuan mengimplementasikan prinsip *Zero Trust Architecture* (ZTA) untuk memperkuat mekanisme otorisasi *OAuth 2.0* pada sistem *REST API* berbasis *microservices*. Pendekatan ZTA menolak kepercayaan implisit dan mewajibkan verifikasi berkelanjutan pada setiap permintaan akses, baik dari dalam maupun luar jaringan. Penelitian ini menggunakan metode kualitatif dengan pendekatan eksperimental melalui tahapan kajian literatur, perancangan sistem, implementasi, pengumpulan data, dan pengujian. Studi kasus dilakukan pada pengembangan sistem keamanan *REST API* berbasis *Spring Boot* dengan *Keycloak* sebagai penyedia identitas dan akses. Penguatan keamanan diterapkan melalui integrasi *Multi-Factor Authentication* (MFA), validasi token *JSON Web Token* (JWT), pemanfaatan *code verifier* sekali pakai, serta Redis sebagai penyimpanan sementara untuk mendukung proses verifikasi dinamis. Pengujian dilakukan dengan membandingkan kondisi sebelum dan sesudah penerapan ZTA berdasarkan parameter skor risiko, *impact*, *likelihood*, dan waktu proses permintaan. Hasil pengujian menunjukkan bahwa penerapan ZTA menurunkan skor risiko rata-rata dari 34,08 menjadi 10,88 atau sebesar 68,1 persen, tanpa menimbulkan dampak signifikan terhadap performa sistem. Waktu proses permintaan tetap berada pada kisaran 7 milidetik dengan perbedaan yang sangat kecil setelah penguatan keamanan diterapkan. Temuan ini membuktikan bahwa ZTA efektif dalam memperkuat otorisasi *OAuth 2.0* pada lingkungan *microservices*, sekaligus menekan risiko serangan seperti *token hijacking*, *brute-force*, *replay attack*, dan *man-in-the-middle* (MiTM). Dengan demikian, penerapan ZTA dapat menjadi pendekatan keamanan yang relevan dan praktis untuk pengembangan *REST API* yang membutuhkan perlindungan data tinggi.

Kata kunci: *Zero Trust Architecture*, *OAuth 2.0*, *REST API*, Keamanan, *Microservices*

Enhancing REST API Security Through Zero Trust Architecture And Keycloak Integration In Microservices-Based Infrastructures

Abstract

The security of REST API services in microservices architecture has become a critical issue as the complexity of digital systems increases and the attack surface expands. This study aims to implement the principles of Zero Trust Architecture (ZTA) to strengthen the OAuth 2.0 authorization mechanism in a microservices-based REST API system. The ZTA approach rejects implicit trust and mandates continuous verification for every access request, whether originating from inside or outside the network. This study employs a qualitative method with an experimental approach, conducted through the following stages: literature review, system design, implementation, data collection, and testing. The case study was carried out on the development of a REST API security system built with Spring Boot, using Keycloak as the identity and access provider. Security hardening was applied through the integration of Multi-Factor Authentication (MFA), JSON Web Token (JWT) validation, the use of a one-time code verifier, and Redis as temporary storage to support dynamic verification processes. Testing was conducted by comparing conditions before and after the implementation of ZTA based on the parameters of risk score, impact, likelihood, and request processing time. The test results indicate that the implementation of ZTA reduced the average risk score from 34.08 to 10.88, representing a decrease of 68.1 percent, without causing any significant impact on system performance. The request processing time remained at approximately 7 milliseconds, with only a marginal difference observed after security hardening was applied. These findings demonstrate that ZTA is effective in strengthening OAuth 2.0 authorization within a microservices environment, while simultaneously mitigating the risk of attacks such as token hijacking, brute-

force, replay attacks, and man-in-the-middle (MiTM) attacks. Therefore, the adoption of ZTA can serve as a relevant and practical security approach for REST API development that requires a high level of data protection.
Keywords: *Zero Trust Architecture, OAuth 2.0, REST API, Security, Microservices*

1. PENDAHULUAN

Pada era transformasi digital saat ini, banyak organisasi beralih dari arsitektur monolitik ke arsitektur *microservices* untuk mendukung inovasi dan efisiensi operasional. Arsitektur *microservices* menawarkan fleksibilitas dan skalabilitas dengan memecah aplikasi besar menjadi layanan-layanan kecil yang saling berinteraksi melalui *Representational State Transfer Application Programming Interface* atau yang dikenal dengan *REST API*. Pertukaran data pada *REST API* umumnya menggunakan format *Javascript Object Notation* (JSON) yang sederhana dan efisien.

Namun, di balik keunggulannya, arsitektur ini memperluas permukaan serangan karena banyaknya titik komunikasi antarlayanan. Kondisi tersebut meningkatkan potensi serangan siber seperti *man-in-the-middle* (MiTM), *injection attacks*, maupun penyalahgunaan identitas pengguna. Selama ini, banyak sistem *REST API* menggunakan mekanisme otorisasi berbasis *OAuth 2.0* untuk mengatur akses ke sumber daya. Namun, kompleksitas sistem terdistribusi modern menuntut pendekatan keamanan yang lebih menyeluruh agar otorisasi tetap terjamin meskipun terjadi kompromi pada salah satu layanan.

Penelitian ini berfokus pada implementasi prinsip *Zero Trust Architecture* (ZTA) untuk memperkuat proses otorisasi berbasis *OAuth 2.0* pada layanan *REST API*. Studi kasus dilakukan menggunakan *Keycloak* sebagai sistem manajemen identitas dan akses, yang mendukung otentikasi berlapis serta integrasi dengan sistem berbasis *Spring Boot*. Dengan pendekatan ini, diharapkan sistem *REST API* mampu mendeteksi, memverifikasi, dan membatasi setiap permintaan akses secara adaptif, sehingga risiko kebocoran data dan eskalasi serangan dapat diminimalkan secara signifikan.

2. TINJAUAN PUSTAKA

Microservices adalah gaya arsitektur di mana komponen dalam suatu sistem dirancang sebagai layanan yang dapat dilakukan *deploy* secara independen. Setiap layanan merepresentasikan subdomain bisnis tertentu dan berkomunikasi menggunakan protokol ringan seperti *HTTP* (Peralta, 2023). *REST API* memungkinkan pertukaran data dengan format *JSON* yang sederhana, efisien, dan mudah dibaca oleh berbagai platform pemrograman (Setyawan dan Syuhada, 2023). Dalam konteks *microservices*, *REST API* menjadi antarmuka utama yang menghubungkan antar layanan maupun antara layanan dengan klien eksternal (Wiguna dan Desnanjaya, 2023).

Arsitektur *microservices* meningkatkan permukaan serangan karena banyaknya titik komunikasi antar layanan, termasuk serangan seperti MiTM dan *injection attacks* (Febriansyah dan Awangga, 2023). Setiap layanan yang berjalan secara independen harus diamankan secara individual, sehingga tantangan otentikasi dan otorisasi menjadi lebih kompleks dibandingkan arsitektur monolitik (de Almeida dan Canedo, 2022). Mekanisme berbasis token seperti JWT menjadi solusi umum untuk menangani otorisasi antar layanan karena bersifat *stateless* dan dapat diverifikasi tanpa bergantung pada sesi terpusat (Venčkauskas et al., 2023). *OAuth 2.0* merupakan standar protokol otorisasi yang memberikan cara aman bagi aplikasi pihak ketiga untuk mengakses data pengguna tanpa membagikan kredensial langsung. Protokol ini dirancang untuk menangani kebutuhan otorisasi modern di berbagai jenis aplikasi dengan memisahkan peran antara pemilik sumber daya, klien, dan *server* otorisasi (Chatterjee dan Prinz, 2022).

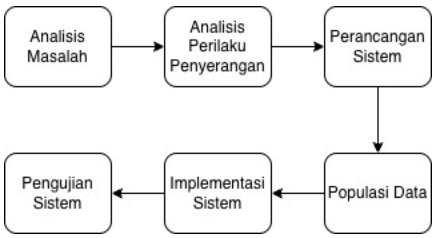
ZTA menolak konsep kepercayaan implisit pada jaringan internal dengan menerapkan validasi berlapis dalam setiap permintaan akses. Prinsip utamanya adalah "jangan percaya, selalu verifikasi", di mana setiap entitas, baik dari dalam maupun luar jaringan, harus melalui proses otentikasi dan otorisasi yang ketat (Fernandez dan Brazhuk, 2024). ZTA diimplementasikan melalui tiga pendekatan utama, yaitu peningkatan tata kelola identitas berbasis kebijakan, mikro-segmentasi jaringan, serta penerapan *software-defined* perimeter (Syed et al., 2022). Dalam lingkungan *cloud* dan infrastruktur yang terdistribusi, ZTA menjadi kerangka keamanan yang relevan karena tidak bergantung pada batas perimeter jaringan tradisional yang semakin tidak efektif (Sarkar et al., 2022). Pendekatan ini dapat memperkuat lapisan keamanan *OAuth 2.0*. melalui integrasi mekanisme seperti verifikasi identitas adaptif, segmentasi akses, dan penerapan *multi-factor authentication* (MFA) (Kang et al., 2023). Dengan menggabungkan prinsip ZTA pada arsitektur *microservices*, organisasi dapat memastikan bahwa setiap permintaan layanan divalidasi secara kontinu, sehingga risiko penyalahgunaan akses dari dalam jaringan dapat diminimalkan (Ahmadi, 2024).

Keycloak adalah platform sumber terbuka untuk *Identity and Access Management* yang mendukung otentikasi dan otorisasi pada aplikasi modern melalui integrasi dengan protokol seperti *OpenID Connect* dan

OAuth 2.0 (Thorgersen dan Silva, 2021). Sebagai solusi *Identity and Access management* (IAM), *Keycloak* menyediakan fitur seperti *Single Sign-On* (SSO), manajemen pengguna terpusat, *identity brokering*, dan otentikasi multi-faktor yang dapat dikonfigurasi sesuai kebutuhan aplikasi (Dimitrijević et al., 2024). *Keycloak* mendukung integrasi dengan berbagai sistem modern, termasuk aplikasi berbasis *Spring Boot*, di mana *Spring Security* dapat dikonfigurasi sebagai *resource server* yang memvalidasi token JWT yang diterbitkan oleh *Keycloak* untuk melindungi *endpoint REST API* dari akses tidak sah (Chatterjee dan Prinz, 2022).

3. METODE PENELITIAN

Penelitian ini menggunakan metode kualitatif dengan pendekatan eksperimental. Tahapan penelitian meliputi analisis masalah, analisis perilaku penyerangan, perancangan sistem otorisasi menggunakan *OAuth 2.0*, penguatan keamanan melalui penerapan prinsip ZTA, pengumpulan data pengguna, serta tahapan pengujian. Kerangka kerja pengembangan sistem digambarkan pada Gambar 1.



Gambar 1. Tahap Penelitian

3.1. Analisis Masalah

Pada tahap ini, penulis menganalisis karakteristik komponen yang memengaruhi keamanan *REST API*. *REST API* digunakan untuk menyediakan akses data antara klien dan server melalui parameter *query*, *request body*, atau *header* (Wiguna dan Desnanjaya, 2023). Dalam arsitektur *microservices*, setiap layanan mengekspos *endpoint* yang dapat diakses secara independen, sehingga jumlah titik masuk yang berpotensi menjadi celah keamanan meningkat secara signifikan (de Almeida dan Canedo, 2022). Tanpa validasi input yang memadai, sistem menjadi rentan terhadap serangan seperti *SQL Injection* atau *Command Injection* yang dapat mengubah, menghapus, atau mencuri data penting (Febriansyah dan Awangga, 2023). Kondisi ini menyebabkan risiko kebocoran data dan menurunnya keandalan layanan. Lebih lanjut, API yang tidak dilindungi dengan mekanisme otorisasi yang tepat juga rentan terhadap akses tidak sah yang memanfaatkan token atau kredensial yang bocor, terutama pada lingkungan berbasis *cloud* yang bersifat dinamis (Gupta, 2025).

3.2. Analisis Perilaku Penyerangan

Analisis ini dilakukan untuk memahami pola serangan yang umum terjadi pada *REST API*. Pelaku serangan biasanya memanfaatkan *endpoint* yang tidak divalidasi dengan baik untuk menyisipkan kode berbahaya, melakukan pengintaian struktur *Application Programming Interface* (API), serta mengeksploitasi data sensitif (Febriansyah dan Awangga, 2023). Selain itu, serangan berbasis token seperti *token hijacking* dan *replay attack* juga umum terjadi ketika mekanisme validasi token tidak diterapkan secara ketat pada setiap sesi komunikasi (Venčkauskas et al., 2023). Pada sistem berbasis *cloud*, aktor ancaman juga kerap memanfaatkan kelemahan konfigurasi akses untuk melakukan *privilege escalation*, yaitu peningkatan hak akses secara ilegal melampaui izin yang seharusnya diberikan (Vasa, 2025). Pemahaman perilaku ini menjadi dasar dalam merancang sistem pertahanan adaptif berbasis ZTA.

3.3. Perancangan Sistem

Sistem dirancang dengan mengintegrasikan otorisasi *OAuth 2.0* yang diperkuat oleh prinsip ZTA. Pada tahap awal, sistem hanya menggunakan *OAuth 2.0* sebagai model dasar otorisasi dengan validasi token berbentuk *JSON Web Token* (JWT). JWT dipilih karena bersifat *stateless* dan memungkinkan verifikasi identitas dilakukan secara lokal oleh setiap layanan tanpa bergantung pada penyimpanan sesi terpusat, sehingga cocok untuk arsitektur *microservices* yang terdistribusi (Venčkauskas et al., 2023). Setelah itu, diterapkan ZTA sebagai lapisan keamanan tambahan melalui mekanisme MFA dan validasi dinamis terhadap setiap permintaan akses.

Prinsip ZTA yang diterapkan mencakup ketentuan sebagai berikut:

1. Verifikasi identitas berkelanjutan.
2. Prinsip hak akses minimum (*least privilege*).
3. Segmentasi jaringan dan sumber daya.
4. Pemantauan aktivitas akses dan audit log.

Keempat prinsip tersebut selaras dengan kerangka ZTA yang dirumuskan oleh Syed et al. (2022), yang menyatakan bahwa implementasi ZTA yang efektif harus mencakup kontrol identitas berbasis kebijakan, mikro-segmentasi, serta otomasi pemantauan keamanan secara berkelanjutan. Penerapan prinsip *least privilege* secara khusus bertujuan memastikan bahwa setiap pengguna atau layanan hanya memiliki hak akses minimum yang diperlukan untuk menjalankan fungsinya, sehingga dampak dari potensi pelanggaran keamanan dapat diminimalkan (Fernandez dan Brazhuk, 2024). Dalam konteks

lingkungan berbasis *cloud* dan *Internet of Things* (IoT), ZTA juga terbukti efektif dalam membatasi pergerakan lateral penyerang yang telah berhasil menembus satu titik akses dalam sistem (Federici et al., 2023). Dengan desain ini, setiap permintaan data yang telah melewati otorisasi *OAuth 2.0* tetap harus diverifikasi ulang oleh sistem berbasis ZTA sebelum akses diberikan, sehingga membentuk lapisan pertahanan berlapis yang adaptif terhadap ancaman siber yang terus berkembang (Weinberg dan Cohen, 2024).

3.4. Populasi Data

Data penelitian berupa data pengguna dan peran (*role*) yang diatur dalam sistem *Keycloak*. *Keycloak* digunakan untuk mengelola identitas pengguna, otentikasi, dan otorisasi secara otomatis tanpa perlu melakukan pemetaan manual (Thorgersen dan Silva, 2021). Manajemen pengguna diintegrasikan dengan sistem eksternal seperti *Lightweight Directory Access Protocol* (LDAP) atau *Active Directory* , yang memungkinkan sinkronisasi data pengguna dari direktori organisasi secara langsung ke dalam *Keycloak* (Dimitrijević et al., 2024). Selain itu, *Keycloak* juga menyediakan layanan SSO dan manajemen sesi yang memperkuat keamanan akses pada lingkungan aplikasi modern (Prinz et al., 2024).

3.5. Implementasi Sistem

Sistem dikembangkan menggunakan bahasa pemrograman *Java* dengan framework *Spring Boot*. *Spring Boot* dipilih karena menyediakan modul *Spring Security* yang dapat dikonfigurasi sebagai *OAuth 2.0 resource server*, sehingga setiap *endpoint REST API* dapat dilindungi melalui validasi token JWT yang diterbitkan oleh *Keycloak* secara otomatis (Chatterjee dan Prinz, 2022). *Keycloak* digunakan sebagai penyedia layanan identitas dan otorisasi, dengan sistem dijalankan pada lingkungan *Docker* yang terintegrasi dengan basis data *PostgreSQL*. Lapisan pengamanan ZTA diterapkan dengan menambahkan modul verifikasi MFA dan validasi token melalui *cache server* yang dalam hal ini menggunakan *Redis* untuk memastikan setiap permintaan akses benar-benar sah.

3.6. Pengujian Sistem

Pengujian dilakukan dengan membandingkan kondisi sistem sebelum dan sesudah penerapan prinsip ZTA pada mekanisme *OAuth 2.0*. Fokus pengujian adalah tingkat keamanan, respons sistem, dan skor risiko dari berbagai skenario akses. Pendekatan pengujian komparatif ini dipilih karena dapat menggambarkan secara kuantitatif sejauh mana penerapan ZTA berkontribusi terhadap penurunan tingkat risiko keamanan pada sistem (Federici et al., 2023). Skenario pengujian dilakukan menggunakan media *Web* dan *Testing Tool* seperti *Postman* atau *Swagger*, meliputi:

1. Proses login dan verifikasi token.
2. Pertukaran data dengan otorisasi *OAuth 2.0* (*baseline*).
3. Pertukaran data setelah penguatan ZTA (*post-implementation*).

Penilaian risiko dilakukan berdasarkan dua parameter utama, yaitu *impact* dan *likelihood*. *Impact* mengukur tingkat keparahan kerugian jika ancaman terjadi, dengan skala 1–10. *Likelihood* mengukur kemungkinan terjadinya ancaman, juga dengan skala 1–10 seperti pada *impact*.

Skor risiko total dihitung untuk menunjukkan efektivitas penerapan ZTA terhadap penurunan potensi serangan, yang kemudian digunakan untuk mengukur peningkatan keamanan dalam satuan persentase.

$$\text{Skor Risiko} = \text{Impact} \times \text{Likelihood}$$

Gambar 2. Perhitungan Skor Risiko

Setelah diperoleh nilai skor risiko, dapat dihitung persentase peningkatan keamanan yang menggambarkan selisih antara skor awal (sebelum penerapan ZTA) dan skor akhir (setelah penerapan ZTA).

$$\text{Peningkatan Keamanan (\%)} = \frac{\text{Skor Awal} - \text{Skor Akhir}}{\text{Skor Awal}} \times 100$$

Gambar 3. Perhitungan Persentase Peningkatan Keamanan

4. HASIL DAN PEMBAHASAN

Penelitian ini menghasilkan sistem *REST API* yang diimplementasikan dengan mekanisme otorisasi *OAuth 2.0* dan lapisan keamanan tambahan melalui penerapan prinsip ZTA. Pengujian dilakukan untuk menilai perubahan tingkat keamanan, konsumsi sumber daya, serta performa sistem sebelum dan sesudah penerapan ZTA. Tujuan pengujian adalah untuk mengetahui efektivitas ZTA dalam memperkuat proses otorisasi *OAuth 2.0*, serta mengidentifikasi dampaknya terhadap performa sistem.

4.1. Konfigurasi Lingkungan Pengujian

Pengujian dilakukan pada lingkungan terkontrol dengan spesifikasi perangkat keras dan perangkat lunak yang konsisten untuk memastikan validitas dan reliabilitas hasil. Skenario pengujian dilakukan pada sepuluh pengguna dengan dua peran berbeda, yaitu *Admin* dan *User*, yang memiliki tingkat hak akses berbeda. Pengguna dengan peran *Admin* memiliki akses penuh terhadap metode *HTTP* seperti *GET*, *POST*, *PUT*, *PATCH*, dan *DELETE*, sedangkan pengguna dengan peran *User* hanya memiliki hak

akses terbatas pada metode *GET* saja. Pembagian peran ini dimaksudkan untuk menguji efektivitas kontrol akses berbasis peran atau *Role-Based Access Control* (RBAC) dalam arsitektur ZTA.

Tabel 1. Sumber Daya Pengujian

Kategori	Spesifikasi
Prosesor	Intel Core i7-11800H (atau setara)
RAM	16 GB DDR4
Penyimpanan	1 TB SSD
Sistem Operasi	Windows 11 Pro

Pemilihan spesifikasi perangkat keras di atas didasarkan pada kebutuhan untuk menjalankan beberapa layanan secara bersamaan dalam arsitektur *microservices*, termasuk *Keycloak*, basis data *PostgreSQL*, *cache server Redis*, serta aplikasi berbasis *Spring Boot*. Kapasitas RAM 16 GB memungkinkan sistem untuk menangani beban kerja yang cukup intensif tanpa mengalami *bottleneck* pada memori, sementara prosesor *multi-core* mendukung pemrosesan paralel yang diperlukan dalam arsitektur terdistribusi.

Sistem dibangun menggunakan lingkungan *microservices* dengan beberapa layanan mandiri yang berinteraksi melalui *REST API*. Setiap layanan memiliki tanggung jawab spesifik dan berkomunikasi melalui protokol *HTTP* dengan format data *JSON*. Komponen utama yang digunakan dalam penelitian ini dapat dilihat pada Tabel 2.

Tabel 2. Komponen dan Alat Pengujian

Kategori	Alat	Kegunaan
Sistem IAM	*Keycloak*	Platform *Identity and Access Management* (IAM)
Framework	*Spring Boot* (*JDK 17*)	Pengembangan aplikasi berbasis *Java*
Basis Data	*PostgreSQL*	Penyimpanan pengguna, pengaturan, dan log audit
Cache Server	*Redis*	Penyimpanan *code verifier* untuk validasi MFA
Server	*Docker*	Operasional server *Keycloak* dan aplikasi *web*
Monitoring Tools	*Elasticsearch* dan *Kibana*	Pemantauan performa (*CPU*, *RAM*, latensi API)
Testing Tools	*Postman* dan *Swagger*	Pengujian dan validasi permintaan API

Penggunaan *Docker* sebagai platform kontainerisasi memungkinkan setiap layanan untuk berjalan secara terisolasi, sehingga memudahkan pengelolaan dan deployment. *Keycloak* dipilih sebagai sistem IAM karena mendukung standar *OAuth 2.0*, *OpenID Connect*, serta memiliki fleksibilitas tinggi dalam konfigurasi kebijakan keamanan. Sementara itu, *Redis* digunakan sebagai *cache server* untuk menyimpan *code verifier* yang dihasilkan secara dinamis dalam proses MFA, dengan tujuan mempercepat validasi tanpa perlu mengakses basis data utama.

4.2. Arsitektur Sistem dan Desain Pengujian

Skenario pengujian dibagi menjadi dua tahap utama yang mencerminkan kondisi sistem sebelum dan sesudah penerapan prinsip ZTA:

1. Sebelum penerapan ZTA – Sistem hanya mengandalkan otorisasi *OAuth 2.0* dengan validasi token berbasis JWT. Pada tahap ini, setiap permintaan API hanya diverifikasi melalui keberadaan dan validitas *access token* yang disertakan dalam *header* permintaan. Jika token valid dan belum kadaluarsa (*expired*), maka permintaan akan diproses tanpa verifikasi tambahan.
2. Sesudah penerapan ZTA – Sistem dilengkapi dengan verifikasi tambahan berbasis MFA dan validasi berlapis pada setiap permintaan data. Selain validasi token, sistem memerlukan *code verifier* yang dihasilkan secara dinamis dan hanya dapat digunakan satu kali. Mekanisme ini memastikan bahwa meskipun *access token* berhasil dicuri atau disalahgunakan, penyerang tidak dapat mengakses sumber daya tanpa memiliki *code verifier* yang valid.

Desain pengujian ini memungkinkan peneliti untuk mengukur dampak langsung dari penerapan ZTA terhadap tingkat keamanan sistem, serta mengidentifikasi *trade-off* antara peningkatan keamanan dan performa sistem.

4.3. Hasil Pengujian Sebelum Penerapan ZTA

Pengujian awal dilakukan terhadap sistem dengan konfigurasi dasar *OAuth 2.0* tanpa penerapan prinsip ZTA. Pada tahap ini, sistem hanya melakukan validasi terhadap *access token* yang disertakan dalam setiap permintaan. Hasil pengujian ditunjukkan pada Tabel 3.

Tabel 3. Hasil Pengujian Sistem Sebelum ZTA

Peran	Media	Waktu (milidetik)	Impact (%)	Likelihood (%)	Skor Risiko (%)
Admin	*Web*	7,45	5,33	3,5	21
Admin	*Testing*	7,29	6,17	5,83	37,5

Peran	Media	Waktu (milidetik)	Impact (%)	Likelihood (%)	Skor Risiko (%)
	Tool				
User	*Web*	7,07	6,17	5,17	33
User	*Testing Tool*	6,84	6,17	7,33	44,83

Dari hasil pengujian pada Tabel 3, dapat diamati bahwa sistem dengan *OAuth 2.0* saja menunjukkan tingkat risiko yang masih berada pada kategori moderat. Meskipun validasi token sudah dilakukan, masih terdapat celah keamanan yang memungkinkan terjadinya gangguan seperti duplikasi token, serangan *brute-force*, atau penyalahgunaan token yang masih valid. Nilai skor risiko rata-rata mencapai 34,08%, yang mengindikasikan bahwa sistem belum sepenuhnya mampu mencegah akses tidak sah, terutama dalam skenario di mana token berhasil dicuri melalui serangan MiTM atau *phishing*.

Perbedaan skor risiko antara pengujian melalui *Web* dan *Testing Tool* menunjukkan bahwa pengujian melalui *Testing Tool* cenderung memiliki nilai *likelihood* yang lebih tinggi. Hal ini disebabkan oleh karakteristik *Postman* atau *Swagger* sebagai alat pengujian API yang memungkinkan otomatisasi permintaan dalam jumlah besar, sehingga meningkatkan potensi serangan *brute-force*. Pada pengujian melalui antarmuka *Web*, terdapat lapisan keamanan tambahan yang disediakan oleh *browser*, seperti *Same-Origin Policy* dan *Content Security Policy*, yang sedikit mengurangi risiko serangan.

Secara performa, respons sistem masih stabil dengan rata-rata waktu proses sekitar 7 milidetik per permintaan. Waktu proses yang relatif cepat ini menunjukkan bahwa validasi *OAuth 2.0* tidak memberikan beban signifikan terhadap kinerja sistem. Namun, kecepatan ini tidak dapat dijadikan jaminan keamanan, karena sistem masih rentan terhadap berbagai jenis serangan yang menargetkan kelemahan pada mekanisme otorisasi berbasis token.

Hasil pengujian juga menunjukkan bahwa pengguna dengan peran *User* memiliki skor risiko yang sedikit lebih tinggi dibandingkan dengan peran *Admin*. Perbedaan ini disebabkan oleh fakta bahwa pengguna dengan peran *User* memiliki akses yang lebih terbatas, sehingga ketika terjadi upaya akses ke sumber daya yang tidak diizinkan, sistem langsung menolak permintaan tersebut tanpa melanjutkan proses validasi lebih lanjut. Sebaliknya, pengguna dengan peran *Admin* memiliki akses penuh, sehingga lebih banyak permintaan yang berhasil diproses, yang pada gilirannya mengurangi nilai *likelihood* dari perspektif pengujian keamanan.

4.4. Analisis Kelemahan OAuth 2.0 Tanpa ZTA

Meskipun *OAuth 2.0* merupakan standar industri untuk otorisasi, implementasi tanpa lapisan keamanan tambahan masih memiliki beberapa kelemahan mendasar. Pertama, sistem hanya mengandalkan validitas *access token* tanpa memverifikasi konteks permintaan, seperti lokasi geografis, perangkat yang digunakan, atau pola akses pengguna. Hal ini memungkinkan penyerang yang berhasil mendapatkan token valid untuk mengakses sistem tanpa terdeteksi.

Kedua, *access token* yang telah diterbitkan tidak dapat dicabut secara langsung hingga masa berlakunya habis (*expired*). Jika token dengan masa berlaku panjang berhasil dicuri, penyerang memiliki jendela waktu yang cukup luas untuk melakukan eksploitasi. Meskipun *refresh token* dapat dicabut, mekanisme ini tidak berlaku untuk *access token* yang sudah diterbitkan.

Ketiga, tidak ada mekanisme untuk mendeteksi penggunaan token yang tidak wajar, seperti penggunaan token yang sama dari beberapa lokasi atau perangkat yang berbeda dalam waktu singkat. Ketiadaan pemantauan kontekstual ini membuat sistem tidak mampu mengidentifikasi aktivitas mencurigakan secara *near-real time*.

4.5. Hasil Pengujian Setelah Penerapan ZTA

Setelah prinsip ZTA diterapkan, sistem melakukan pemeriksaan tambahan terhadap setiap permintaan. Selain validasi token, sistem kini menggunakan *code verifier* melalui modul MFA yang secara dinamis memverifikasi identitas pengguna. *Code verifier* ini berupa string acak sepanjang 26 karakter yang terdiri dari kombinasi huruf besar, huruf kecil, dan angka, yang dihasilkan setiap kali pengguna akan melakukan permintaan data. Kode ini kemudian disimpan sementara dalam *cache server Redis* dengan masa berlaku terbatas, sehingga memastikan Tingkat keamanan yang lebih tinggi dan mencegah penggunaan ulang.

Tabel 4. Hasil Pengujian Sistem Setelah Penerapan ZTA

Peran	Media	Waktu (milidetik)	Impact (%)	Likelihood (%)	Skor Risiko (%)
Admin	*Web*	7,55	3,67	1,5	6,17
Admin	*Testing Tool*	6,78	4,67	2,67	13
User	*Web*	7,03	4,17	2,5	10,83
User	*Testing Tool*	6,82	4,17	3,33	13,5

Setelah penerapan ZTA, terjadi penurunan signifikan pada skor risiko menjadi rata-rata 10,88%. Penurunan sebesar 68,1% dari kondisi sebelum ZTA

menunjukkan bahwa verifikasi berlapis berhasil mengurangi potensi kebocoran data dan serangan berbasis *token reuse*. Penurunan nilai *impact* dan *likelihood* secara bersamaan mengindikasikan bahwa tidak hanya kemungkinan terjadinya serangan yang berkurang, tetapi juga dampak yang ditimbulkan jika serangan berhasil dilakukan.

Dari sisi performa, peningkatan lapisan keamanan hanya menambah waktu pemrosesan rata-rata sekitar 0,3 milidetik, yang masih tergolong memberikan dampak latensi yang minimal. Penambahan waktu ini disebabkan oleh proses validasi *code verifier* yang melibatkan akses ke *cache server Redis*. Meskipun ada penambahan latensi, nilai ini masih dalam batas yang dapat diterima untuk sistem yang berorientasi pada keamanan tinggi, terutama untuk layanan yang menangani data sensitif.

Hasil pengujian juga menunjukkan bahwa pengujian melalui *Testing Tool* mengalami tingkat kegagalan yang lebih tinggi setelah penerapan ZTA. Hal ini disebabkan oleh fakta bahwa *code verifier* hanya dapat dihasilkan dan digunakan melalui alur yang sah, yaitu dari antarmuka *Web* yang terintegrasi dengan sistem. Mekanisme ini secara efektif memblokir upaya otomatisasi serangan atau penggunaan alat pengujian eksternal yang tidak memiliki akses ke *code verifier* yang valid. Dengan demikian, ZTA berhasil menambahkan lapisan pertahanan yang tidak hanya bergantung pada validitas token, tetapi juga pada konteks dan mekanisme pembangkitan permintaan.

4.6. Mekanisme Keamanan dalam ZTA

Penerapan ZTA dalam penelitian ini melibatkan beberapa mekanisme keamanan yang bekerja secara sinergis untuk memperkuat otorisasi *OAuth 2.0*. Pertama adalah verifikasi identitas berkelanjutan, di mana setiap permintaan diperlakukan sebagai permintaan baru yang harus diverifikasi secara independen, tanpa asumsi kepercayaan berdasarkan verifikasi sebelumnya.

Kedua adalah prinsip hak akses minimum, yang memastikan bahwa setiap pengguna hanya memiliki akses ke sumber daya yang benar-benar diperlukan untuk menjalankan tugasnya. Dalam implementasi ini, pengguna dengan peran *User* hanya dapat melakukan operasi baca (*GET*), sementara operasi tulis, ubah, dan hapus (*POST*, *PUT*, *PATCH*, *DELETE*) hanya dapat dilakukan oleh pengguna dengan peran *Admin*.

Ketiga adalah validasi berbasis MFA, di mana sistem memerlukan faktor verifikasi tambahan selain *access token*. *Code verifier* yang dihasilkan secara dinamis dan disimpan di *cache server Redis* dengan masa berlaku singkat (biasanya 60-120 detik) berfungsi sebagai faktor verifikasi kedua. Setelah *code verifier* digunakan satu kali, kode tersebut langsung dihapus dari *cache*, sehingga tidak dapat digunakan kembali meskipun masih dalam masa berlaku.

Keempat adalah segmentasi jaringan dan sumber daya, di mana setiap layanan dalam arsitektur *microservices* diisolasi dan hanya dapat diakses melalui jalur komunikasi yang telah ditentukan. Hal ini mencegah pergerakan lateral penyerang dalam sistem, sehingga meskipun satu layanan berhasil dikompromi, penyerang tidak dapat dengan mudah mengakses layanan lainnya.

4.7. Analisis Perbandingan Penerapan ZTA

Penilaian efektivitas penerapan ZTA secara kuantitatif, dilakukan perbandingan skor rata-rata sistem pada dua kondisi tersebut.

Tabel 5. Perbandingan Hasil Pengujian

Kondisi Sistem	Skor Risiko Rata-Rata (%)	Waktu Proses Rata-Rata (milidetik)	Peningkatan Keamanan (%)
Sebelum ZTA (*OAuth 2.0* saja)	34,08	7,16	-
Setelah ZTA (*OAuth 2.0* + ZTA)	10,88	7,05	68,1

Berdasarkan hasil perbandingan Tabel 5, penerapan ZTA mampu menurunkan skor risiko hingga sekitar 68,1% dengan penurunan waktu proses rata-rata sebesar 0,11 milidetik. Hasil yang menunjukkan ini menunjukkan bahwa penerapan ZTA tidak hanya meningkatkan keamanan, tetapi juga sedikit meningkatkan efisiensi sistem dalam beberapa skenario. Peningkatan efisiensi ini disebabkan oleh fakta bahwa banyak permintaan tidak valid atau tidak sah yang langsung ditolak pada tahap awal verifikasi, sehingga tidak perlu melanjutkan ke proses pengolahan data yang lebih kompleks.

Peningkatan keamanan sebesar 68,1% ini disebabkan oleh tiga faktor utama:

1. Verifikasi identitas adaptif yang mencegah penggunaan ulang token. Dengan adanya *code verifier*, penyerang yang berhasil mendapatkan *access token* valid tetap tidak dapat mengakses sistem tanpa memiliki *code verifier* yang sesuai.
2. Pemisahan hak akses berdasarkan peran pengguna. Prinsip ini memastikan bahwa setiap pengguna hanya dapat mengakses sumber daya yang relevan dengan perannya, sehingga membatasi dampak jika akun pengguna berhasil dikompromi.
3. Validasi berbasis *cache server* (*Redis*) yang memastikan setiap *code verifier* hanya dapat digunakan satu kali. Mekanisme *one-time use* ini

sangat efektif dalam mencegah serangan *replay attack*, di mana penyerang mencoba menggunakan kembali permintaan yang telah berhasil dilakukan sebelumnya.

Hasil pengujian menunjukkan bahwa peran *Admin* memiliki waktu proses yang sedikit lebih tinggi dibanding *User* pada pengujian melalui *Web*, namun sedikit lebih cepat pada pengujian melalui *Testing Tool*. Perbedaan ini disebabkan oleh kompleksitas hak akses yang lebih banyak pada peran *Admin* dan validasi tambahan yang dilakukan oleh sistem pada setiap permintaan data. Pada pengujian melalui *Testing Tool*, banyak permintaan dari peran *User* yang langsung ditolak karena tidak memiliki *code verifier* yang valid, sehingga waktu pemrosesan menjadi lebih singkat.

4.8. Interpretasi Hasil dan Implikasi Praktis

Secara keseluruhan, integrasi prinsip ZTA pada sistem *OAuth 2.0* memberikan peningkatan keamanan yang signifikan tanpa mengorbankan kinerja sistem secara berarti. Pengujian menunjukkan bahwa setiap permintaan akses kini harus melalui dua tahap otorisasi yaitu validasi token dan verifikasi MFA melalui *code verifier*. Pendekatan berlapis ini berhasil memperkecil kemungkinan ancaman seperti *brute-force attack*, *token hijacking*, dan MiTM.

Penurunan nilai *impact* dan *likelihood* yang signifikan membuktikan bahwa ZTA berperan efektif dalam memperkuat mekanisme otorisasi berbasis *OAuth 2.0* pada arsitektur *REST API*. Dari perspektif praktis, hasil penelitian ini menunjukkan bahwa organisasi yang mengadopsi arsitektur *microservices* dapat meningkatkan keamanan sistem mereka secara substansial dengan menerapkan prinsip ZTA, tanpa perlu melakukan perubahan radikal pada infrastruktur yang sudah ada.

Implikasi praktis dari temuan ini mencakup beberapa aspek. Pertama, organisasi dapat mengimplementasikan ZTA secara bertahap, dimulai dari layanan yang menangani data paling sensitif, kemudian diperluas ke layanan lainnya. Kedua, biaya implementasi relatif terjangkau karena sebagian besar komponen yang diperlukan (seperti *Keycloak*, *Redis*, dan *Docker*) tersedia sebagai perangkat lunak sumber terbuka. Ketiga, peningkatan keamanan yang diperoleh sebanding dengan investasi yang diperlukan, baik dari segi waktu maupun sumber daya.

Dari perspektif pengguna akhir, penerapan ZTA dengan MFA mungkin menambah sedikit kompleksitas dalam proses akses ke sistem. Namun, penelitian ini menunjukkan bahwa penambahan waktu proses sangat minimal (rata-rata hanya 0,3 milidetik), sehingga tidak berdampak signifikan terhadap pengalaman pengguna. Dengan komunikasi yang baik mengenai manfaat keamanan yang diperoleh, pengguna umumnya akan menerima sedikit peningkatan kompleksitas sebagai *trade-off* yang wajar untuk perlindungan data yang lebih baik.

5. KESIMPULAN DAN SARAN

5.1. Kesimpulan

Berdasarkan hasil pengujian dan analisis yang telah dilakukan, penerapan prinsip ZTA terbukti mampu memperkuat mekanisme otorisasi *OAuth 2.0* pada layanan *REST API*. Integrasi prinsip ZTA, seperti verifikasi berlapis melalui MFA dan validasi identitas yang berkelanjutan, secara signifikan menurunkan tingkat risiko keamanan tanpa memberikan dampak berarti terhadap performa sistem.

Rata-rata waktu proses permintaan data meningkat dalam skala yang kecil akibat adanya tahapan validasi tambahan, namun peningkatan ini masih dalam batas wajar untuk sistem yang berorientasi pada keamanan tinggi. Penerapan ZTA berhasil membatasi akses tidak sah, mencegah serangan seperti MiTM dan *brute-force*, serta memastikan setiap permintaan akses diverifikasi sesuai dengan identitas dan hak pengguna.

Dengan demikian, penelitian ini menunjukkan bahwa penerapan prinsip ZTA pada sistem *OAuth 2.0* dapat meningkatkan efektivitas kontrol akses dan resiliensi sistem tanpa mengorbankan efisiensi kinerja. Hasil ini dapat menjadi dasar bagi pengembangan standar keamanan berlapis pada ekosistem *microservices* di masa mendatang.

5.2. Saran

Penelitian selanjutnya disarankan untuk mengembangkan model penerapan prinsip ZTA pada berbagai skenario sistem *REST API* yang lebih kompleks, seperti integrasi dengan layanan cloud dan arsitektur *hybrid*. Studi lanjutan dapat mengkaji efektivitas ZTA dalam menghadapi serangan tipe baru atau memperluas fitur *adaptive security*. Selain itu, optimalisasi mekanisme validasi berlapis dan evaluasi performa pada volume data serta jumlah pengguna yang lebih besar akan memperkaya temuan dan memperkuat standar keamanan pada ekosistem *microservices*.

DAFTAR PUSTAKA

Ahmadi, S. (2024) ‘Zero Trust Architecture in Cloud Networks: Application, Challenges and Future Opportunities’, *Journal of Engineering Research and Reports*, 26(2), pp. 215–228. doi:10.9734/jerr/2024/v26i21083.

Ali, M. et al. (2026) ‘Zero Trust Architecture in Cloud-Native Environments: A Scalable Security

Approach', *International Journal of Science and Research Archive*, 18(1), pp. 296–305. doi:10.30574/ijsra.2026.18.1.0063.

de Almeida, M.G. and Canedo, E.D. (2022) 'Authentication and Authorization in Microservices Architecture: A Systematic Literature Review', *Applied Sciences*, 12(6), p. 3023. doi:10.3390/app12063023.

Febriansyah, F. and Awangga, R.M. (2023) *Membangun RESTful API dengan Go*. Bandung: Penerbit Buku Pedia.

Federici, F. et al, V. (2023) 'A Zero-Trust Architecture for Remote Access in Industrial IoT Infrastructures', *Electronics*, 12(3), p. 566. doi:10.3390/electronics12030566.

Fernandez, B. and Brazhuk, A. (2024) 'A Critical Analysis of Zero Trust Architecture (ZTA)', *Computer Standards & Interfaces*, p. 103832. doi:10.1016/j.csi.2024.103832.

Gupta, R.K. (2025) 'Zero-trust architecture as a framework for cloud API security', *World Journal of Advanced Research and Reviews*, 26(1), pp. 3389–3398. doi:10.30574/wjarr.2025.26.1.1446.

Hendra et al. (2025) 'Memperkuat Autentikasi dan Integritas Data REST-API Menggunakan Token HMAC SHA-256', *Jurnal Minfo Polgan*, 13(2), pp. 2189–2197. Available at: https://jurnal.polgan.ac.id/index.php/jmp/article/view/14406.

Kang, H. et al. (2023) 'Theory and Application of Zero Trust Security: A Brief Survey', *Entropy*, 25(12), p. 1595. doi:10.3390/e25121595.

Kusnanto, Y. et al. (2024) 'Implementasi Zero Trust Architecture untuk Meningkatkan Keamanan Jaringan', *Jurnal STKIP PGRI Tulungagung*, 6(2). Available at: https://jurnal.stkippgritulungagung.ac.id/index.php/jipi/article/view/6943.

Li, Y. et al. (2022) 'Role-Based Access Control Model for Inter-System Cross-Domain in Multi-Domain Environment', *Applied Sciences*, 12(24), p. 13036. doi:10.3390/app122413036.

Mpamugo, E. and Ansa, G. (2024) 'Enhancing Network Security in Mobile Applications with Role-Based Access Control', *Journal of Information Systems and Informatics*, 6(3), pp. 1872–1899. doi:10.51519/journalisi.v6i3.863.

Peralta, J.H. (2023) *Microservice APIs Using Python, Flask, FastAPI, OpenAPI and More*. Birmingham: Manning Publications.

Prinz, C. et al. (2024) 'JWT, OAuth, LDAP and Keycloak in API Security Frameworks', *CEUR Workshop Proceedings*, 3676. Available at: https://ceurspt.wikidata.dbis.rwth-aachen.de/Vol-3676/short_09.pdf.

Rose, L. et al. (2022) 'Applying Spring Security Framework with KeyCloak-Based OAuth2 to Protect Microservice Architecture APIs: A Case Study', *Sensors*, 22(5), p. 1703. doi:10.3390/s22051703.

Sarkar, S. et al. (2022) 'Security of Zero Trust Networks in Cloud Computing: A Comparative Review', *Sustainability*, 14(18), p. 11213. doi:10.3390/su141811213.

Setyawan, M.Y.H. and Syuhada, E.G. (2023) *Pengembangan Dashboard Laporan Bulanan untuk Monitoring Kinerja Perusahaan*. Bandung: Penerbit Buku Pedia.

Syed, N.F. et al. (2022) 'Zero Trust Architecture (ZTA): A Comprehensive Survey', *IEEE Access*, 10, pp. 57143–57179. doi:10.1109/ACCESS.2022.3174679.

Thorgersen, S. and Silva, P.I. (2021) *Keycloak – Identity and Access Management for Modern Applications*. Birmingham: Packt Publishing.

Vasa, R. (2025) 'ZERO-TRUST SECURITY IN CLOUD API INTEGRATIONS FOR HEALTHCARE SYSTEMS', *TPM: Testing, Psychometrics, Methodology in Applied Psychology*, 32(S8), pp. 467–475. Available at: https://tpmap.org/submission/index.php/tpm/article/view/2644.

Venčkauskas, A et al. (2023) 'Enhancing Microservices Security with Token-Based Access Control Method', *Sensors*, 23(6), p. 3363. doi:10.3390/s23063363.

Weinberg, A.I. and Cohen, K. (2024) 'Zero Trust Implementation in the Emerging Technologies Era: A Survey', *Complex Engineering Systems*, 4, p. 16. doi:10.20517/ces.2024.41.

Wiguna, I.K.A.G. and Desnanjaya, I.G.M.N. (2023) *Konsep API dan Implementasinya dalam Membangun Sistem Informasi Menggunakan Laravel*. Denpasar: PT Sonpedia Publishing Indonesia.