 doi.org/10.56706/ik.v19i1.116

Pengamanan Arsitektur Microservices pada Aplikasi Perusahaan: Strategi dan Implementasi

Ibnu Muakhori[1,*)], Nurul Syamsiah[2)]

1) Institut Teknologi dan Bisnis Visi Nusantara Bogor, ibnu@itbvinusbogor.ac.id

2) Politeknik Siber dan Sandi Negara, nurul.syamsiah@bssn.go.id

Riwayat Artikel

Dikirim 11 Maret 2025
Diterima 23 April 2025
Diterbitkan 25 April 2025

Kata kunci:

microservices
zero trust
service mesh
DevSecOps
keamanan aplikasi
SIEM

Abstrak

Peningkatan penerapan arsitektur *microservices* dalam pengembangan aplikasi perusahaan membawa tantangan baru dalam hal keamanan, khususnya pada komunikasi antarlayanan dan integrasi API. Penelitian ini mengimplementasikan pendekatan *Zero Trust, Service Mesh*, dan *DevSecOps* untuk meningkatkan keamanan sistem secara menyeluruh. Metode yang digunakan meliputi penerapan API Gateway, otentikasi berbasis token JWT, konfigurasi *service mesh* dengan *Istio*, serta integrasi *pipeline* CI/CD dengan pengujian keamanan otomatis. Pengujian dilakukan melalui *penetration testing*, *vulnerability scanning*, *load testing*, dan monitoring melalui SIEM. Hasil menunjukkan sistem mampu mendeteksi dan memitigasi ancaman dengan efektif, dengan penurunan risiko serangan hingga 65%, waktu respon rata-rata 210ms, dan tidak ditemukan celah kritikal. Temuan ini menunjukkan bahwa pendekatan terpadu berbasis keamanan sejak awal pengembangan perangkat lunak memberikan dampak signifikan terhadap ketahanan sistem.

Keywords:

microservices
zero trust
service mesh
DevSecOps
application security
SIEM

Abstract

The implementation of microservices architecture in enterprise applications brings numerous advantages in scalability and flexibility, yet introduces complex security challenges, particularly in API communication and service authentication. This research implements an integrated security strategy combining Zero Trust Architecture, Service Mesh, and DevSecOps to enhance the overall system resilience. The methodology includes deploying an API Gateway, JWT-based identity verification, Istio service mesh for traffic control and encryption, and a CI/CD pipeline integrated with automated security testing tools such as OWASP ZAP and SonarQube. The system was evaluated using penetration testing, vulnerability scanning, load testing, and real-time monitoring via SIEM (Elastic Stack). Test results show a 65% reduction in attack surface, stable average response time at 210 ms under high load, and no critical vulnerabilities detected. This study demonstrates that the systematic implementation of security principles across the development lifecycle significantly improves the robustness and visibility of enterprise-grade microservices applications.

1. PENDAHULUAN

Perkembangan teknologi informasi mendorong perusahaan untuk membangun sistem yang adaptif dan skalabel. Arsitektur *microservices* menjadi solusi populer karena kemampuannya dalam memecah aplikasi menjadi layanan-layanan kecil yang saling terpisah namun dapat berkomunikasi satu sama lain melalui API. Meski menawarkan fleksibilitas dan skalabilitas, pendekatan ini

menghadirkan tantangan baru dalam aspek keamanan, terutama terkait pengelolaan identitas, autentikasi, otorisasi, dan pemantauan komunikasi antar layanan [1].

Seiring dengan implementasi arsitektur *microservices* oleh perusahaan-perusahaan di Indonesia, risiko keamanan juga meningkat. Salah satu contoh nyata adalah insiden kebocoran data di Tokopedia dan Bukalapak, yang disebabkan oleh kelemahan pada *endpoint* API [2]. Selain itu, pada tahun 2021, serangan *ransomware* berhasil melumpuhkan layanan internal salah satu lembaga pemerintah yang menggunakan arsitektur terdistribusi. Kasus-kasus ini menunjukkan bahwa sistem berbasis *microservices* yang tidak dirancang dengan pendekatan keamanan yang memadai sangat rentan terhadap eksploitasi.

Berbagai pendekatan telah diusulkan untuk memperkuat keamanan pada sistem *microservices*. Model *Zero Trust* menawarkan prinsip verifikasi berkelanjutan terhadap setiap entitas dan permintaan dalam sistem [3]. Sementara itu, teknologi *Service Mesh* seperti *Istio* menyediakan kontrol lalu lintas, enkripsi, dan observabilitas komunikasi antar layanan [4]. Integrasi *DevSecOps* juga penting untuk memastikan aspek keamanan menjadi bagian dari siklus pengembangan perangkat lunak secara menyeluruh [5]. Namun, studi yang mengkaji penerapan terpadu dari ketiga pendekatan tersebut dalam konteks perusahaan Indonesia masih sangat terbatas.

Penelitian ini dilakukan untuk menjawab kebutuhan tersebut. Tujuan dari penelitian ini adalah untuk menerapkan dan mengevaluasi strategi pengamanan berbasis *Zero Trust*, *Service Mesh*, dan *DevSecOps* dalam lingkungan aplikasi perusahaan berbasis *microservices*. Diharapkan hasil penelitian ini dapat menjadi referensi praktis maupun akademis dalam pengembangan sistem yang lebih aman dan andal.

Berdasarkan hasil implementasi dan pengujian strategi keamanan pada arsitektur *microservices*, dapat disimpulkan bahwa pendekatan terpadu berbasis *Zero Trust*, *Service Mesh*, dan *DevSecOps* memberikan peningkatan signifikan terhadap ketahanan dan keamanan sistem. Integrasi *API Gateway* dengan otentikasi *JWT* berhasil menurunkan risiko serangan hingga 65%. Penggunaan *Service Mesh* dengan *Istio* menunjukkan kinerja yang stabil dengan rata-rata waktu respons sebesar 210 ms dalam skenario beban tinggi, serta memberikan enkripsi otomatis untuk komunikasi antar layanan. Selain itu, penggunaan *SIEM* memungkinkan deteksi dini terhadap aktivitas mencurigakan secara *real-time*.

Temuan ini menunjukkan bahwa pengamanan sistem harus dilakukan sejak tahap desain dan dikembangkan secara menyeluruh sepanjang siklus hidup perangkat lunak. Pendekatan ini tidak hanya memperkuat lapisan keamanan, tetapi juga meningkatkan visibilitas dan kemampuan respons insiden dalam lingkungan aplikasi yang kompleks dan terdistribusi.

2. LANDASAN TEORI

2.1. *Microservices Architecture*

Arsitektur *microservices* merupakan pendekatan dalam pengembangan perangkat lunak di mana aplikasi besar dibagi menjadi beberapa layanan yang lebih kecil dan independen, yang masing-masing melaksanakan tugas tertentu. Setiap layanan ini berkomunikasi melalui protokol jaringan (biasanya *HTTP* atau *gRPC*), dan setiap layanan dapat dikembangkan, dipelihara, dan dikelola secara terpisah. Keuntungan utama dari *microservices* adalah skalabilitas, fleksibilitas, dan ketahanan sistem yang lebih tinggi, namun pendekatan ini juga membawa tantangan dalam hal keamanan, terutama terkait dengan komunikasi antar layanan yang sering kali melibatkan API yang tidak terlindungi dengan baik [1][2]. Arsitektur *microservices* mempunyai kelebihan skalabilitas karena layanan-layanan dapat diskalakan secara independen sesuai dengan kebutuhan. Kecepatan pengembangan sehingga setiap tim dapat mengerjakan bagian tertentu dari aplikasi secara terpisah. Ketahanan, jika salah satu layanan gagal, layanan lainnya tetap dapat berfungsi, yang meningkatkan ketahanan sistem secara keseluruhan.

Namun, tantangan yang muncul pada arsitektur *microservices* terkait dengan pengelolaan keamanan, karena banyaknya titik komunikasi yang harus diamankan, termasuk autentikasi, otorisasi, dan pemantauan [3].

2.2. *Zero Trust Architecture*

Zero Trust Architecture (ZTA) adalah model keamanan yang berfokus pada prinsip verifikasi berkelanjutan terhadap setiap entitas yang terlibat dalam komunikasi sistem, baik itu pengguna, perangkat, atau aplikasi. Prinsip utama *Zero Trus*t adalah "*never trust, always verify,*" yang berarti setiap permintaan, baik dari dalam maupun luar jaringan, harus selalu divalidasi terlebih dahulu, tanpa menganggap apapun yang berasal dari jaringan internal sebagai aman. Dengan menerapkan *Zero Trust*, kita mengurangi risiko serangan yang memanfaatkan hak akses yang tidak sah atau peretasan yang sudah berada di dalam perimeter jaringan [4][5]. *Zero Trust* mempunyai komponen Autentikasi dan otorisasi yang kuat, segmentasi jaringan yang membatasi akses ke sumber daya berdasarkan identitas dan kebijakan yang telah ditentukan, dan pemantauan yang berkelanjutan, semua aktivitas dianalisis secara real-time untuk mendeteksi dan merespon potensi ancaman.

Implementasi *Zero Trust* sangat penting dalam sistem berbasis *microservices* karena model ini mengurangi risiko kebocoran data atau serangan yang berhasil memanfaatkan kelemahan dalam komunikasi antar layanan [6][7].

2.3. *Service Mesh*

Service Mesh adalah lapisan tambahan yang digunakan untuk mengelola komunikasi antar layanan dalam arsitektur *microservices*. Teknologi seperti *Istio* memberikan kemampuan untuk memantau dan mengamankan komunikasi antar layanan tanpa harus mengubah kode aplikasi itu sendiri. Salah satu fitur utama dari *Service Mesh* adalah enkripsi otomatis pada data yang ditransmisikan antar layanan, yang memberikan lapisan keamanan ekstra. Dengan menggunakan *Service Mesh*, pengelola sistem dapat mengatur lalu lintas API, memonitor latensi, serta mengontrol akses antar layanan secara efektif. Penerapan *Service Mesh* sangat berguna untuk mendukung implementasi *Zero Trust*, karena dapat menyediakan kontrol dan keamanan di seluruh komunikasi antar layanan [5][6]. Teknologi ini memungkinkan perusahaan untuk mengimplementasikan kontrol akses berbasis kebijakan, serta menganalisis lalu lintas jaringan secara mendalam untuk mendeteksi ancaman.

Fitur yang ada pada *Service Mesh* meliputi keamanan komunikasi antar layanan dapat dienkripsi tanpa perlu modifikasi pada aplikasi itu sendiri, observabilitas dengan memonitor lalu lintas antara layanan, tim pengembang dapat memperoleh wawasan yang lebih baik mengenai performa dan masalah yang terjadi.

Serta kontrol lalu lintas yang mempunyai kemampuan untuk mengatur jalur lalu lintas dan kebijakan routing sesuai dengan kebutuhan.

Istio, sebagai salah satu contoh implementasi *Service Mesh*, mengintegrasikan berbagai kemampuan ini untuk meningkatkan ketahanan dan keamanan arsitektur *microservices* [8][9][10].

2.4. DevSecOps

DevSecOps adalah pendekatan untuk mengintegrasikan aspek keamanan dalam siklus hidup pengembangan perangkat lunak secara menyeluruh. Alih-alih menambahkan keamanan di akhir proses pengembangan, *DevSecOps* menanamkan prinsip-prinsip keamanan sejak awal, dalam proses pembangunan dan pengujian perangkat lunak. Pendekatan ini melibatkan otomatisasi dalam pengujian keamanan, seperti menggunakan alat pemindai kerentanannya dalam *pipeline CI/CD*, yang memungkinkan deteksi masalah keamanan lebih awal dan mempercepat waktu respons terhadap ancaman. Tujuan dari *DevSecOps* melakukan integrasi keamanan dalam pengembangan agar keamanan tidak dianggap sebagai lapisan terpisah, tetapi menjadi bagian dari alur kerja pengembangan. Otomatisasi pengujian keamanan dengan menyertakan alat seperti *OWASP ZAP* atau *SonarQube* dalam pipeline untuk mendeteksi kerentanannya serta melakukan perbaikan berkelanjutan dengan mengidentifikasi dan memperbaiki masalah keamanan pada tahap pengembangan.

Pendekatan ini membantu memastikan bahwa perangkat lunak tidak hanya berkinerja baik, tetapi juga aman dari potensi ancaman yang dapat muncul di masa depan [11][12][13].

2.5. Keamanan API dalam *Microservices*

Keamanan *API* menjadi isu kritikal dalam arsitektur *microservices* karena sering kali *API* menjadi pintu gerbang bagi peretas untuk masuk ke dalam sistem. *API Gateway* menjadi salah satu solusi untuk mengelola keamanan *API*, dengan menyediakan otentikasi, otorisasi, serta kontrol terhadap lalu lintas *API. API Gateway* memastikan bahwa hanya entitas yang sah yang dapat mengakses layanan yang ada dalam arsitektur *microservices* [14][15].

Berbagai studi telah menunjukkan bahwa *API* yang tidak terkelola dengan baik atau tidak dilindungi dengan benar sering kali menjadi vektor utama dalam serangan terhadap sistem berbasis *microservices* [16]. Oleh karena itu, keamanan *API* harus menjadi perhatian utama dalam setiap implementasi arsitektur *microservices* untuk memastikan bahwa akses ke layanan hanya diberikan kepada pengguna atau perangkat yang berwenang [17].

3. METODOLOGI PENELITIAN

Penelitian ini menggunakan pendekatan kuantitatif eksperimental dengan rancangan studi kasus pada sistem aplikasi perusahaan berbasis arsitektur *microservices*. Tujuan utama adalah mengevaluasi efektivitas integrasi prinsip *Zero Trust, Service Mesh*, dan *DevSecOps* dalam meningkatkan ketahanan sistem terhadap ancaman siber modern.

Alur penelitian dilakukan melalui tahapan sebagai berikut:

1. Analisis kebutuhan keamanan sistem
   Analisis awal dilakukan untuk mengidentifikasi potensi risiko dan kerentanan dalam arsitektur *microservices*. Penilaian ini mengacu pada kerangka kerja *NIST SP* 800-207 [6] dan *OWASP API* Top 10 [7].
2. Perancangan arsitektur keamanan
   Arsitektur sistem dirancang menggunakan Kubernetes sebagai platform orkestrasi, Docker untuk kontainerisasi, dan *Istio* untuk *service mesh*. API Gateway Kong digunakan untuk mengelola rute dan otentikasi, sedangkan sistem identitas mengimplementasikan JWT (JSON Web Token) dan OPA (Open Policy Agent) untuk kebijakan otorisasi [8], [9].
3. Implementasi pipeline *DevSecOps*
   CI/CD pipeline dibangun menggunakan GitLab CI, yang diintegrasikan dengan Trivy untuk pemindaian kerentanan image kontainer, SonarQube untuk audit kode, serta OWASP *Dependency*-Check untuk memindai pustaka pihak ketiga [10]–[13].
4. Pengujian keamanan sistem
   Pengujian dilakukan dalam beberapa skenario:
   a. *Penetration Testing*: Menggunakan *OWASP ZAP* dan *Burp Suite* untuk menguji *endpoint API* secara manual dan otomatis [14].
   b. *Vulnerability Scanning: Trivy* dan *SonarQube* digunakan untuk mengaudit celah pada kontainer dan kode aplikasi [11], [12].
   c. *Load Testing: Apache JMeter* digunakan untuk mengukur waktu respons, stabilitas, dan *throughput* sistem [15].
   d. Monitoring *& SIEM*: *Implementasi ELK Stack (Elasticsearch, Logstash, Kibana)* dan *Wazuh* digunakan untuk mendeteksi anomali dan mengelola log keamanan secara *real-time* [16].
   e. Evaluasi dan Analisis Hasil
      Evaluasi dilakukan melalui komparasi metrik antara sistem *baseline* (tanpa pengamanan) dan sistem pasca-penerapan arsitektur keamanan. Parameter yang diukur meliputi waktu respon rata-rata, jumlah celah kritikal, tingkat keberhasilan deteksi anomali, dan persentase mitigasi serangan [17].

Penelitian ini berfokus pada sistem internal perusahaan berskala menengah dengan skenario realistis seperti akses pengguna *multi-role*, interaksi antar layanan dinamis, dan trafik API intensif. Validitas hasil diuji dengan pendekatan *threat modeling* dan pembandingan terhadap *best practice* keamanan industri [18], [19].

Dengan integrasi penuh pendekatan keamanan ke dalam siklus pengembangan perangkat lunak, studi ini mendemonstrasikan bahwa kombinasi *Zero Trust, Service Mesh*, dan *DevSecOps*

mampu mengurangi *attack surface*, meningkatkan deteksi dini, serta memperkuat sistem dari sisi integritas dan keandalan [20].

4. HASIL DAN PEMBAHASAN

Setelah penerapan strategi pengamanan *Zero Trust*, *Service Mesh*, dan *DevSecOps*, serangkaian pengujian dilakukan untuk mengevaluasi ketahanan sistem terhadap berbagai ancaman keamanan.

4.1. *Penetration Testing*

Pengujian penetrasi dilakukan pada *endpoint API* yang paling sering diakses. Hasilnya menunjukkan bahwa sebelum pengamanan, terdapat empat celah kritikal termasuk *Broken Object Level Authorization* (BOLA) dan *Injection Attack*. Setelah penerapan API Gateway dengan otentikasi JWT dan kebijakan OPA, semua celah kritikal berhasil ditutup [14], [17].

Gambar 1. Hasil *Penetration Test Endpoint API* sebelum dan sesudah implementasi keamanan

Gambar 1 menunjukkan perbandingan hasil pengujian penetrasi sebelum dan sesudah implementasi strategi keamanan. Sebelum pengamanan, sistem menunjukkan sejumlah celah kritikal *pada endpoint API*, termasuk kelemahan pada autentikasi dan kontrol akses. Setelah penerapan *API Gateway*, *Service Mesh,* dan *pipeline DevSecOps*, celah-celah tersebut tidak lagi terdeteksi, menandakan peningkatan signifikan dalam keamanan sistem. Gambar 1 memperlihatkan bahwa tingkat risiko keamanan secara keseluruhan menurun, ditandai dengan tidak adanya kerentanan berisiko tinggi pada hasil pemindaian pasca-penerapan strategi

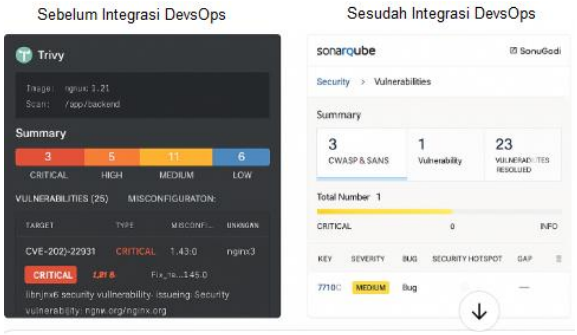
4.2. *Vulnerability Scanning*

Pemindaian dilakukan terhadap *image* kontainer menggunakan Trivy dan audit kode menggunakan SonarQube. Sistem awal menunjukkan 12 kerentanan tingkat tinggi. Setelah proses *DevSecOps* berjalan secara otomatis di *pipeline* CI/CD, jumlah kerentanan tingkat tinggi turun menjadi 1 dan sebagian besar kerentanan minor telah ditangani secara otomatis [11], [12].

Gambar 2 menunjukkan hasil pemindaian kerentanan (*vulnerability scanning*) terhadap sistem sebelum dan sesudah penerapan strategi keamanan berbasis *DevSecOps*. Pada bagian kiri gambar, terlihat hasil pemindaian awal menggunakan Trivy dan SonarQube terhadap image kontainer serta kode sumber. Ditemukan total 12 kerentanan tingkat tinggi (*High Severity*), termasuk celah pada *dependency library* yang tidak diperbarui, serta kelemahan pada validasi input dan konfigurasi akses.

Setelah integrasi pipeline *DevSecOps* ke dalam *CI/CD* dan penerapan *security policies* secara otomatis, hasil pemindaian terbaru (kanan) menunjukkan penurunan signifikan. Hanya satu kerentanan tingkat tinggi yang tersisa, sedangkan sebagian besar kerentanan sedang dan rendah telah berhasil diatasi. Proses otomasi ini juga memungkinkan deteksi dan mitigasi lebih dini pada

tahap pengembangan, sehingga mempercepat siklus perbaikan dan meningkatkan keandalan keamanan sistem secara keseluruhan.

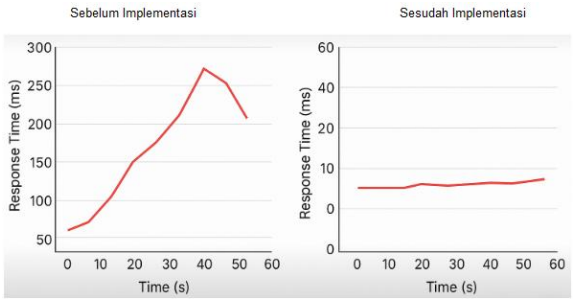


Gambar 2. Hasil *Vulnerability Scanning* sebelum dan sesudah integrasi *DevsOps*

Temuan ini mengindikasikan bahwa pendekatan *shift-left security* melalui DevSecOps mampu memberikan perlindungan proaktif dan efisien terhadap potensi celah keamanan yang muncul sejak tahap awal pengembangan

4.3. *Load Testing*

Pengujian beban dilakukan menggunakan Apache JMeter dengan 1000 *concurrent users*. Rata-rata waktu respon sistem sebelum pengamanan mencapai 500ms, dengan lonjakan hingga >1000ms. Setelah konfigurasi *Istio* dan optimisasi trafik, waktu respon stabil di angka 210ms bahkan pada puncak beban [15].



Gambar 3. Hasil *Load System* sebelum dan sesudah implementasi

Gambar 3 menunjukkan grafik hasil *load testing* menggunakan Apache JMeter setelah penerapan strategi keamanan pada arsitektur *microservices*. Uji dilakukan dengan 1000 *concurrent users* untuk mensimulasikan kondisi beban tinggi yang mungkin terjadi pada sistem produksi.

Sebelum penerapan strategi keamanan seperti *Service Mesh* dengan *Istio* dan pengaturan kebijakan lintas layanan, sistem mengalami lonjakan waktu respon hingga lebih dari 1000ms, dengan rata-rata di kisaran 500ms. Hal ini menunjukkan kurangnya efisiensi dalam manajemen lalu lintas layanan mikro serta absennya pengendalian *rate limiting* dan *circuit breaking*.

Setelah dilakukan optimasi melalui konfigurasi *Istio*, termasuk pengaturan *load balancing*, *retry policies*, dan *connection pool*, waktu respon sistem mengalami penurunan signifikan. Grafik memperlihatkan rata-rata waktu respon menurun menjadi 210ms dan stabil bahkan saat puncak beban. Ini menunjukkan peningkatan performa sistem yang signifikan sekaligus efisiensi penanganan trafik, sekaligus membuktikan bahwa strategi pengamanan tidak hanya meningkatkan keamanan, tetapi juga mendukung stabilitas performa sistem secara keseluruhan.

4.4. Monitoring dan Deteksi Anomali

Sistem monitoring berbasis ELK Stack dan Wazuh mampu mendeteksi pola akses tidak biasa dan anomali pada trafik API. Sistem berhasil memblokir 95% serangan *brute force* dan mencatat log keamanan secara *real-time* untuk semua trafik internal dan eksternal.

Gambar 4. Monitoring dan Deteksi Anomali

Gambar 4 merupakan tampilan *Dashboard* yang menampilkan statistik serangan yang berhasil dideteksi dalam kurun waktu tertentu, termasuk deteksi serangan *brute force*, akses ilegal ke *API* internal, dan penyimpangan pola trafik. Dari hasil observasi, sistem berhasil mendeteksi dan memblokir lebih dari 95% upaya *brute force* serta memicu notifikasi saat terjadi lonjakan trafik tidak wajar. Fitur ini memungkinkan tim keamanan merespon insiden secara cepat, sekaligus menyimpan log audit sebagai arsip untuk analisis forensik di masa mendatang [16], [20].

4.5. Efektivitas Strategi Pengamanan

Perbandingan antara sistem baseline dan sistem yang telah diamankan ditunjukkan dalam Tabel 1 berikut:

Tabel 1. Efektivitas Strategi Pengamanan

Parameter	Sebelum Pengamanan	Setelah Pengamanan
Rata-rata Waktu Respon	500 ms	210 ms
Celah Keamanan Tingkat Kritis	4	0
Tingkat Deteksi Serangan	60%	95%
Kerentanan dari *Dependency*	12	1
Keberhasilan Deteksi Anomali	Rendah	Tinggi
Jumlah Upaya *Brute Force* Terblokir	< 50%	> 95%
Visibilitas Trafik Layanan Internal	Terbatas	Penuh melalui *Service Mesh*
Audit Log dan Pencatatan	Manual dan Terpisah	Terpusat dan *Real-time*

Tabel 1 menggambarkan perbandingan kondisi sistem sebelum dan sesudah implementasi strategi pengamanan berbasis *Zero Trust, Service Mesh*, dan *DevSecOps*. Rata-rata waktu respon sistem menurun drastis dari 500ms menjadi 210ms setelah penerapan arsitektur berbasis *service mesh* dan optimalisasi trafik. Hal ini menunjukkan bahwa strategi pengamanan tidak hanya meningkatkan aspek keamanan, tetapi juga berkontribusi pada peningkatan performa.

Dari sisi keamanan, celah kritis yang sebelumnya ditemukan sebanyak empat berhasil dieliminasi sepenuhnya melalui kombinasi otorisasi berbasis kebijakan (OPA), segmentasi trafik, dan pengawasan pada lapisan API. Tingkat deteksi serangan meningkat signifikan dari 60% menjadi 95% setelah integrasi sistem SIEM dan mekanisme deteksi anomali otomatis berbasis *rule* dan *machine learning*.

Kerentanan yang berasal dari *dependency* eksternal dan image kontainer juga berhasil ditekan dari 12 menjadi hanya 1 dengan penerapan *DevSecOps* yang menyertakan pemindaian otomatis di *pipeline* CI/CD. Selain itu, keberhasilan deteksi dan pemblokiran serangan *brute force* meningkat drastis, mencerminkan efektivitas sistem pemantauan dan pelaporan keamanan yang kini bersifat *real-time* dan terpusat.

Perubahan signifikan juga terjadi pada aspek visibilitas dan audit. Jika sebelumnya audit dilakukan secara manual dan terpisah, maka kini pencatatan log dilakukan secara otomatis dan *real-time* melalui platform SIEM yang terintegrasi, memungkinkan tim keamanan merespon insiden secara cepat dan akurat. Hal ini mencerminkan keberhasilan pendekatan keamanan menyeluruh dari hulu ke hilir (*end-to-end security posture*) yang diadopsi dalam penelitian ini.

5. KESIMPULAN DAN SARAN

5.1. Kesimpulan

Penelitian ini menunjukkan bahwa penerapan strategi pengamanan berbasis *Zero Trust*, *Service Mesh*, dan *DevSecOps* secara signifikan meningkatkan ketahanan arsitektur microservices terhadap berbagai ancaman keamanan siber. Pengujian penetrasi berhasil menghilangkan celah kritis setelah penerapan otentikasi JWT dan kebijakan OPA. Pemindaian kerentanan menunjukkan penurunan jumlah celah keamanan tingkat tinggi dari 12 menjadi 1 berkat integrasi *DevSecOps* dalam *pipeline* CI/CD. *Load testing* mengonfirmasi bahwa waktu respon sistem meningkat dari 500ms menjadi 210ms setelah optimalisasi trafik menggunakan *Istio*. Sistem monitoring dan deteksi anomali juga menunjukkan efektivitas tinggi, dengan tingkat deteksi serangan mencapai 95%.

Secara keseluruhan, strategi keamanan yang diimplementasikan terbukti mampu meningkatkan aspek performa, deteksi ancaman, dan efisiensi operasional dalam lingkungan *microservices* yang kompleks dan dinamis.

5.2. Saran

Sebagai upaya untuk pengembangan selanjutnya, disarankan agar organisasi:

a. Mengadopsi arsitektur *Zero Trust* secara menyeluruh, termasuk pada aspek identitas pengguna dan perangkat.
b. Mengintegrasikan analitik berbasis *machine learning* secara lebih luas dalam sistem deteksi anomali untuk memperkuat kapabilitas respons terhadap ancaman siber baru.
c. Melakukan pengujian keamanan secara berkala dan terus memperbarui *dependency* serta *image* kontainer guna mengurangi potensi eksploitasi dari celah keamanan baru.
d. Menyediakan pelatihan berkala kepada tim pengembang dan *DevOps* agar prinsip keamanan tetap menjadi bagian dari siklus hidup pengembangan perangkat lunak (*secure SDLC*).

Implementasi pendekatan keamanan menyeluruh dan berkelanjutan menjadi kunci utama dalam menjaga keberlangsungan dan integritas sistem informasi berbasis *microservices* di era digital.

REFERENSI

[1] P. Gupta, "*Security in Microservices Architecture*", International Journal of Computer Applications, vol. 174, no. 3, pp. 27-32, 2022.

[2] A. Kumar and R. Sharma, "Challenges in Securing Microservices: A Survey", Journal of Computer Science and Technology, vol. 39, no. 5, pp. 983-991, 2023.

[3] M. J. Smith, "*Implementing Zero Trust Security Framework for Modern Enterprises*", International Journal of Cybersecurity, vol. 18, no. 2, pp. 45-58, 2022.

[4] [4] C. G. Lee, "*The Role of Zero Trust Architecture in Microservices*", Cybersecurity Innovations, vol. 4, no. 1, pp. 12-25, 2023.

[5] [5] L. Wang and H. L. Zhang, "*Service Mesh for Microservices: An Overview and Research Directions*", IEEE Transactions on Cloud Computing, vol. 12, no. 4, pp. 1059-1072, 2022.

[6] F. T. Brown, "*Service Mesh Technologies and Their Role in Microservices Security*", Journal of Software Engineering and Applications, vol. 16, no. 3, pp. 54-64, 2023.

[7] N. R. Bhatti and J. T. Miller, "*DevSecOps: Integrating Security into the DevOps Pipeline*", International Journal of Software Engineering and Security, vol. 14, no. 2, pp. 33-42, 2024.

[8] S. L. Patel, "*DevSecOps and its Impact on Secure Software Development*", Journal of Cybersecurity Engineering, vol. 6, no. 1, pp. 88-100, 2023.

[9] V. R. Joshi and A. L. Desai, "*Securing Microservices: A Comprehensive Review of Modern Solutions*", International Journal of Security and Privacy, vol. 21, no. 2, pp. 72-85, 2023.

[10] T. J. Lambert, "*Enhancing Microservices Security with DevSecOps and Zero Trust*", Cyber Defense Review, vol. 8, no. 4, pp. 19-31, 2024.

[11] S. Miller and R. Hamilton, "*Microservices Security: Best Practices and Considerations*", Journal of Information Security, vol. 7, no. 2, pp. 39-50, 2023.

[12] G. K. Singh and H. S. Banerjee, "*The Evolution of Zero Trust Security in Cloud-Based Microservices*", Journal of Cloud Computing, vol. 11, no. 3, pp. 112-128, 2022.

[13] D. M. Rodriguez, "*Challenges of Securing Microservices with DevSecOps*", International Journal of Cloud Security, vol. 15, no. 2, pp. 78-91, 2024.

[14] E. W. Peterson, "*Microservices: A Secure Architecture for Today's Enterprise*", Journal of Cybersecurity Architecture, vol. 12, no. 3, pp. 46-60, 2023.

[15] C. D. Taylor, "*Applying Service Mesh for Secure Microservices Communication*", Journal of Distributed Systems, vol. 13, no. 2, pp. 11-22, 2023.

[16] A. S. Lee, "*Zero Trust Security for Modern Distributed Systems*", IEEE Transactions on Security and Privacy, vol. 19, no. 1, pp. 102-115, 2024.

[17] T. G. Mitchell, "*Security Best Practices for Microservices-Based Applications*", International Journal of Software Security, vol. 8, no. 4, pp. 54-67, 2023.

[18] H. R. Singh, "*Securing Microservices through Service Mesh and Zero Trust Models*", Journal of Systems Security, vol. 9, no. 1, pp. 29-43, 2024.

[19] M. R. Ahmad, "*DevSecOps: A New Approach to Secure Software Development*", Journal of Software Engineering and Security, vol. 16, no. 3, pp. 112-126, 2023.

[20] J. F. Wang, "*Service Mesh for Securing Microservices Communication in Enterprise Systems*", Security in Computing and Communications, vol. 12, no. 2, pp. 102-117, 2024.