# BAB 5
# KESIMPULAN DAN SARAN

## 5.1 KESIMPULAN

Penelitian ini berhasil mengembangkan ekosistem *Static Application Security* (SAST) menggunakan pendekatan DevSecOps untuk mengidentifikasi dan mereduksi kerentanan keamanan pada kode sumber aplikasi. Melalui penerapan *Secure Software Development Life Cycle* (SSDLC), proses penelitian ini diawali dengan Tahap Identifikasi dan Analisis yang melibatkan Penilaian Kerentanan (*Vulnerability Assessment* – VA) menggunakan reNgine untuk mengidentifikasi kelemahan keamanan yang terlihat dari luar, konfigurasi server yang salah, dan kerentanan pada komponen pihak ketiga. Bersamaan dengan itu, *Static Application Security Testing* (SAST) menggunakan SonarQube berperan sentral dalam memindai kode sumber secara statis untuk mengenali anomali, praktik pengkodean yang tidak efisien (*code smells*), serta kelemahan keamanan internal seperti *insecure coding practices*, *hardcoded credentials*, atau *poor input validation*.

Pada Tahap Pengumpulan dan Pengolahan Data, repositori GitHub (https://github.com/0x1Jar/DevSecOps-Demo-TA.git) berfungsi sebagai sumber kode yang disimulasikan untuk analisis kerentanan, di mana SonarQube secara spesifik mengumpulkan dan mengolah data keamanan langsung dari kode sumber tersebut. Sistem kemudian dirancang secara cermat dalam Tahap Perancangan dan Implementasi dengan mempertimbangkan berbagai aspek keamanan, dan alat analisis keamanan seperti SonarQube diintegrasikan ke dalam *pipeline* Jenkins untuk secara otomatis mendeteksi celah keamanan setiap kali ada perubahan kode sebelum aplikasi dipublikasikan. Penggunaan Jenkins sebagai CI/CD *pipeline* yang terintegrasi dengan GitHub untuk repositori kode dan Docker untuk proses *build* dan *deployment* merupakan komponen kunci dalam arsitektur sistem ini.

Dalam Tahap Pengujian dan Evaluasi, pengujian keamanan dilakukan dengan dua pendekatan utama: VA menggunakan reNgine dan SAST menggunakan SonarQube. Temuan awal dari reNgine mengidentifikasi beberapa kerentanan

informasional dan potensial, seperti *WAF Detection* (informasional), *Credentials Disclosure Check* (*UNKNOWN*), dan *Missing Subresource Integrity* (informasional). Sementara itu, hasil analisis SAST dengan SonarQube menemukan sejumlah 23 kerentanan kode yang diklasifikasikan berdasarkan prioritasnya. Ini termasuk kerentanan prioritas tinggi seperti *Authentication* (kredensial *hardcoded*), *Command Injection*, dan *Cross-Site Request Forgery (CSRF)*; prioritas sedang yang mencakup *Code Injection (RCE)*, masalah *Permission* (izin longgar, paparan port tidak perlu, *base image* besar, menyalin file sensitif), serta *Weak Cryptography*; dan prioritas rendah seperti *Insecure Configuration* (*debug mode* aktif di produksi) dan *Others* (skrip eksternal dari sumber tidak aman, *unnecessary packages*, algoritma *hashing* lemah, *information disclosure*).

Temuan paling bermanfaat dari penelitian ini adalah keberhasilan signifikan dalam mereduksi bahkan mengeliminasi seluruh kerentanan yang teridentifikasi setelah implementasi perbaikan kode. Setelah langkah-langkah remediasi diterapkan berdasarkan temuan dari tahap pengujian, pemindaian ulang dilakukan pada Tahap *Deployment* dan *Monitoring*. Hasil pemindaian ulang VA menggunakan reNgine menunjukkan bahwa semua kerentanan yang sebelumnya terdeteksi (*WAF Detection, Credentials Disclosure Check, Missing Subresource Integrity)* kini berstatus "*Closed*". Demikian pula, pemindaian ulang SAST menggunakan SonarQube menunjukkan bahwa seluruh 23 kerentanan kode yang teridentifikasi sebelumnya telah "*Fixed*". Perbandingan antara hasil pemindaian pertama dan kedua ini menjadi indikator kunci keberhasilan strategi DevSecOps yang diimplementasikan, menunjukkan penurunan drastis jumlah dan tingkat keparahan kerentanan setelah perbaikan kode, menegaskan efektivitas siklus manajemen kerentanan dan tujuan SSDLC untuk deteksi dini serta mitigasi kelemahan keamanan telah tercapai.



## 5.2 SARAN

Berdasarkan hasil penelitian yang telah dicapai dan potensi pengembangan lebih lanjut, terdapat beberapa saran untuk meningkatkan efektivitas dan cakupan

keamanan aplikasi di masa mendatang. Meskipun pemindaian kerentanan menggunakan reNgine telah dilakukan sebagai bagian dari *Vulnerability Assessment* (VA) untuk mengidentifikasi kelemahan dari perspektif eksternal, temuan spesifik dari alat bantu seperti Nuclei yang terintegrasi dengan reNgine belum dideskripsikan secara mendetail dalam laporan ini. Hal ini menyebabkan potensi wawasan yang lebih dalam dari hasil pemindaian tersebut belum sepenuhnya tergali atau disajikan secara komprehensif, sehingga analisis terhadap jenis dan sifat kerentanan yang terdeteksi dari sudut pandang eksternal dapat diperkaya.

Penelitian ini masih secara eksklusif berfokus pada pendekatan *Vulnerability Assessment* (VA) dan *Static Application Security Testing* (SAST). Ini berarti analisis kerentanan dilakukan dengan memeriksa kode sumber secara statis tanpa perlu mengeksekusi aplikasi atau melakukan simulasi serangan aktif pada lingkungan *runtime*. Akibatnya, kerentanan yang hanya dapat muncul atau terdeteksi saat aplikasi berjalan (seperti masalah konfigurasi pada saat *deployment*, *session management*, atau *business logic flaws*) belum sepenuhnya tercakup dalam metodologi pengujian ini.

Oleh karena itu, disarankan untuk penelitian selanjutnya dapat:

1. Melakukan analisis yang lebih mendalam dan menyajikan hasil secara rinci dari setiap alat pemindaian yang digunakan dalam reNgine, termasuk Nuclei, untuk memberikan pemahaman yang lebih komprehensif mengenai jenis dan sifat kerentanan yang terdeteksi dari perspektif eksternal. Detail ini akan memperkaya laporan dan membantu mengidentifikasi area perbaikan yang lebih spesifik pada lapisan aplikasi yang terekspos.
2. Mengintegrasikan *Dynamic Application Security Testing* (DAST) atau pengujian penetrasi (penetration testing) secara penuh ke dalam siklus DevSecOps. Dengan menambahkan DAST, penelitian dapat melengkapi kemampuan SAST dengan mendeteksi kerentanan yang hanya muncul saat aplikasi berjalan (*runtime vulnerabilities*). Pendekatan gabungan SAST dan DAST akan memberikan evaluasi keamanan yang lebih holistik, akurat, dan

mencakup celah keamanan dari seluruh siklus hidup aplikasi, dari kode hingga operasional di lingkungan produksi.