BAB 1
PENDAHULUAN

1.1 LATAR BELAKANG

Industri produksi *software* saat ini telah mengalami peningkatan kompleksitas dalam beberapa tahun terakhir (Caniglia dkk., 2025). Untuk memenuhi kebutuhan tersebut, sebagian besar organisasi menerapkan pendekatan *Software Development Life Cycle* (SDLC) sebagai panduan proses pengembangan *software*. Namun, implementasi SDLC yang umum digunakan saat ini sering kali mengabaikan aspek keamanan sejak awal proses pengembangan. Kelemahan utama dalam penerapan SDLC konvensional di antaranya kurangnya perhatian terhadap aspek keamanan informasi selama proses pengembangan aplikasi. Permasalahan utama dalam implementasi SDLC mencakup kurangnya perhatian terhadap aspek kerahasiaan (*confidentiality*), integritas (*integrity*), dan ketersediaan (*availability*) informasi selama pengembangan aplikasi (Putera dkk., 2020). Selain itu, kurangnya penerapan mekanisme autentikasi, otorisasi, dan akuntabilitas secara konsisten menyebabkan aplikasi yang dihasilkan rentan terhadap berbagai ancaman keamanan seperti serangan injeksi, *cross-site scripting* (XSS), serta *broken authentication* (Moegiono, 2023). Kajian tersebut menekankan pentingnya mengembangkan kerangka kerja keamanan yang tangguh dan fleksibel, yang mampu diintegrasikan secara optimal ke dalam berbagai model SDLC, baik yang saat ini digunakan maupun yang akan berkembang di masa mendatang (Resti dkk., 2024).

Secure Software Development Life Cycle (SSDLC) adalah konsep metodologis yang termasuk dalam *Software Development Life Cycle*, yang di dalamnya melakukan *analysis*, *design*, *implementation* (building), *testing*, dan *evaluation* (deployment and maintenance) (Irfansyah dkk., 2024). SSDLC mengintegrasikan praktik keamanan dalam setiap tahap pengembangan aplikasi, mulai dari perencanaan, analisis kebutuhan, desain, implementasi, pengujian, hingga penerapan. Dalam hal ini, proses *Secure Software Development Life Cycle*

PERPUSTAKAAN UNIVERSITAS JENDERAL ACHMAD YANI YOGYAKARTA

(SSDLC) mencakup berbagai praktik dan aktivitas yang berfokus pada aspek keamanan. Oleh karena itu, penerapan langkah-langkah SSDLC secara tepat menjadi hal yang sangat penting untuk memastikan peningkatan keamanan *software* (Resti dkk., 2024).

Penerapan *Secure Software Development Lifecycle* (SSDLC) menjadi perhatian penting dalam berbagai penelitian sebelumnya. Implementasi SSDLC mampu mengatasi kerentanan pada *software* dengan pendekatan yang lebih efektif dibandingkan dengan SDLC konvensional, penerapan ini mampu meminimalkan potensi risiko serangan seperti *Cross Site Scripting* dan *SQL Injection* secara signifikan (Hasan dkk., 2021). Beberapa faktor yang mempengaruhi *software* memiliki celah, di antaranya konfigurasi kode yang salah. Kesalahan tersebut membuat resiko terjadinya *SQL Injection* sebesar 26.38% dan XSS sebesar 35.57%. *Open Web Application Security Project* (OWASP) melaporkan bahwa *SQL injection* tetap menjadi salah satu dari 10 ancaman keamanan aplikasi web yang sering terjadi, sekitar 9% dari semua pelanggaran aplikasi web disebabkan oleh *SQL injection* (Martin Otieno dkk., 2023). Menurut laporan Akamai, XSS menyumbang 30% dari semua serangan aplikasi web yang dilaporkan. Serangan *SQL Injection* dapat menyebabkan kerusakan data yang parah, kehilangan data penting, atau pengambilalihan penuh pada sistem. Hal tersebut tidak hanya dapat mengganggu operasional sistem, menimbulkan *downtime*, dan merusak reputasi organisasi maupun perusahaan. Praktik *Penetration Testing* saat ini menjadi kewajiban yang harus dilakukan sebagian perusahaan, hal tersebut dilakukan untuk meminimalkan kerugian akibat serangan *exploitasi* atau *data breach* (Setiawan & Ghiffari, 2024).

Secara garis besar *Penetration Testing* merupakan metodologi untuk mengidentifikasi kerentanan dalam suatu sistem atau jaringan, serta mengevaluasi bagaimana langkah-langkah keamanan yang telah diadopsi mungkin rentan terhadap eksploitasi (Yaacoub dkk., 2023). *Framework Information Systems Security Assessment Framework* (ISSAF) salah satunya sebagai standar pengujian penetrasi yang digunakan untuk menguji ketahanan situs web, memiliki beberapa keunggulan dibandingkan kontrol keamanan lainnya, dan berfungsi secara

PERPUSTAKAAN UNIVERSITAS JENDERAL ACHMAD YANI YOGYAKARTA

pandangan teknis dan manajerial (Darmayuda, 2021).Penerapan ISSAF dapat diterapkan dalam berbagai bidang dan aspek di antaranya di sektor pendidikan. Sebagai contoh pengujian keamanan sistem informasi IKIP PGRI Madiun yang memiliki kerentanan *default credentials*. Berdasarkan pengujian yang selaras dengan OWASP top ten 2017 menunjukkan kerentanan *broken authentication* (Silmina dkk., 2022).

Dalam pengembangan *Software* sering kali tidak mengimplementasi *secure coding* sehingga berdampak memiliki kerentanan. Untuk mengurangi dampak potensi kerentanan maka diperlukan pengembangan dengan pendekatan teknik *Static Application Security Testing* (SAST). Pada penelitian ini mengimplementasikan *secure coding* menggunakan platform Snyk yang diintegrasikan menggunakan CI/CD Gtihub *action*. Hasil penelitian menunjukkan bahwa menggunakan teknik *secure coding,* 10 situs web Politeknik Caltex Riau ditemukan kerentanan. Situs tersebut, memiliki 80 kerentanan kode dengan kategori *high* yang terdeteksi dari platform synk (Rayhan dkk., 2023). Penting bahwa sebuah *website* mengimplementasi SAST, teknik ini mensimulasikan bagaimana menganalisis *secure code,* kemungkinan terjadinya kerentanan. Meskipun penelitian tersebut cukup efektif namun pada penelitian ini, sebagai langkah untuk menyempurnakan dan menambahkan beberapa *Tool* yang akan digunakan, antara lain SonarQube sebagai analisis kode, Jenkins sebagai CI/CD, Github sebagai *repository* kode dan docker sebagai *server deployment* pada website. Pengujian *Static Application Security Testing (SAST)* diimplementasikan dalam *secure development life-cycle,* di mana SAST menganalisis *source code* secara statis, kemungkinan kerentanan yang bisa terjadi seperti, *in secure coding practices, hardcoded credentials,* atau *poor input validation*. Metode SAST menerapkan simulasi deteksi sebuah kode secara *real time*, memungkinkan *developer* untuk mengatasi *security breache* selama proses *development life-cycle* (Waheed, 2024).

Studi ini menekankan bahwa implementasi *Secure Software Development Life Cycle* (SSDLC) dengan pendekatan *Static Application Security Testing* (SAST) berdampak baik dalam memperkuat keamanan aplikasi web. Penggabungan SonarQube, Jenkins, GitHub, dan Docker dalam *CI/CD pipeline* memungkinkan

PERPUSTAKAAN UNIVERSITAS JENDERAL ACHMAD YANI YOGYAKARTA

identifikasi kerentanan secara langsung selama proses pengembangan. Ada beberapa batasan, termasuk kemungkinan hasil *false positive* dalam SAST dan keterbatasan untuk mendeteksi kerentanan yang muncul saat *runtime* yang hanya bisa diidentifikasi melalui *Dynamic Application Security Testing* (DAST) atau pengujian penetrasi manual. Oleh sebab itu, studi selanjutnya bisa diarahkan pada penggabungan metode SAST dan DAST.

1.2 PERUMUSAN MASALAH

Penelitian ini difokuskan pada pemanfaatan *Static Application Security Testing* (SAST) sebagai komponen inti strategi keamanan dalam *Secure Software Development Lifecycle* (SSDLC) dan *DevSecOps*, yang bertujuan mengidentifikasi celah pada *source code* sejak dini untuk mengurangi risiko eksploitasi. Kajian ini secara spesifik akan mengevaluasi efektivitas penerapan *DevSecOps* dan metode *secure coding* berbasis SAST dalam peningkatan keamanan aplikasi web, serta menganalisis kontribusi integrasi SonarQube, Jenkins, GitHub, dan Docker dalam alur CI/CD terhadap meminimilasi kerentanan kode.

Ruang lingkup penelitian ini dibatasi pada penggunaan SAST (tidak termasuk DAST/IAST), dengan perangkat spesifik meliputi SonarQube, Jenkins, GitHub, dan Docker, serta diimplementasikan pada aplikasi web berukuran kecil hingga menengah, tidak mencakup sistem korporasi besar atau arsitektur *cloud-native*. Penelitian ini memfokuskan pada pemanfaatan *Static Application Security Testing* (SAST) sebagai strategi keamanan inti dalam *Secure Software Development Lifecycle* (SSDLC) dan DevSecOps. Fokus utamanya adalah mengidentifikasi kerentanan pada *source code* sejak fase awal untuk meminimalkan risiko eksploitasi. Oleh karena itu, masalah yang akan dikaji adalah bagaimana mengevaluasi efektivitas penerapan DevSecOps dan metode *secure coding* berbasis SAST dalam meningkatkan keamanan aplikasi *web*, serta menganalisis kontribusi integrasi SonarQube, Jenkins, GitHub, dan Docker dalam alur CI/CD untuk mengurangi kerentanan kode. Meskipun SAST efektif dalam deteksi dini, perlu diakui adanya batasan seperti kemungkinan *false positive* dan keterbatasan dalam mendeteksi kerentanan *runtime*.

PERPUSTAKAAN UNIVERSITAS JENDERAL ACHMAD YANI YOGYAKARTA

1.3 PERTANYAAN PENELITIAN

Berikut ini beberapa pertanyaan yang menjadi landasan penelitian ini:

1. Bagaimana implementasi CI/CD untuk mengintegrasi ke *source code* pada *repository* dengan *Static Application Security Testing* (SAST)?
2. Bagaimana analisis *source code* untuk mengindetifikasi dengan *Static Application Security Testing* (SAST)*?*
3. Bagaimana implementasi *Static Application Security Testing* (SAST) ke fase deployment atau evaluasi dalam ekosistem container?

1.4 TUJUAN PENELITIAN

Mengacu pada manfaat penelitian dan pertanyaan penelitian yang telah disusun, tujuan dari penelitian ini adalah:

1. Mengintegrasikan CI/CD pada repositori untuk mengidentifikasi kerentanan pada *source code* aplikasi *website* menggunakan *Static Application Security Testing* (SAST).
2. Menggunakan SonarQube sebagai alat untuk melakukan pemindaian *source code* pada aplikasi *web* untuk mengidentifikasi kerentanan.
3. Melakukan fase evaluasi dalam ekosistem *container* menggunakan Docker setelah identifikasi kerentanan.

1.5 MANFAAT HASIL PENELITIAN

Adapun manfaat yang dapat diperoleh dari penelitian ini adalah sebagai berikut:

1. Menjadi panduan mengenai Static Application Security Testing (SAST) untuk mengidentifikasi dan mengurangi kerentanan dalam ekosistem DevSecOps.
2. Bagi developer dan perusahaan membantu upaya keamanan aplikasi *website* dalam fase pengembangan dan mengidentifikasi kerentanan *source code* pada aplikasi *website*.

PERPUSTAKAAN UNIVERSITAS JENDERAL ACHMAD YANI YOGYAKARTA