IMPLEMENTASI DEVSECOPS: MEKANISME HARD-BLOCKING PADA PIPELINE CI MENGGUNAKAN SAST SONARQUBE DAN JENKINS

SKRIPSI

IKHSAN FADILLAH 2207421013

PROGRAM STUDI TEKNIK MULTIMEDIA DAN JARINGAN

JURUSAN TEKNIK INFORMATIKA DAN KOMPUTER

POLITEKNIK NEGERI JAKARTA

TAHUN 2026

IMPLEMENTASI DEVSECOPS: MEKANISME HARD-BLOCKING PADA PIPELINE CI MENGGUNAKAN SAST SONARQUBE DAN JENKINS

SKRIPSI

Dibuat untuk Melengkapi Syarat-Syarat yang Diperlukan
untuk Memperoleh Diploma Empat Politeknik

IKHSAN FADILLAH

2207421013

PROGRAM STUDI TEKNIK MULTIMEDIA DAN JARINGAN
JURUSAN TEKNIK INFORMATIKA DAN KOMPUTER
POLITEKNIK NEGERI JAKARTA

2026

SURAT PERNYATAAN BEBAS PLAGIARISME

Saya yang bertanda tangan di bawah ini:

Nama	:	Ikhsan Fadillah
NIM	:	2207421013
Jurusan/Program Studi	:	Teknik Informatika dan Komputer / Teknik Multimedia dan Jaringan
Judul Skripsi	:	IMPLEMENTASI DEVSECOPS: MEKANISME HARD-BLOCKING PADA PIPELINE CI MENGGUNAKAN SAST SONARQUBE DAN JENKINS

Menyatakan dengan sebenarnya bahwa skripsi ini benar-benar merupakan hasil karya saya sendiri, bebas dari peniruan terhadap karya dari orang lain. Kutipan pendapat dan tulisan orang lain ditunjuk sesuai dengan cara-cara penulisan karya ilmiah yang berlaku.

Apabila di kemudian hari terbukti atau dapat dibuktikan bahwa dalam skripsi ini terkandung ciri-ciri plagiat dan bentuk-bentuk peniruan lain yang dianggap melanggar peraturan, maka saya bersedia menerima sanksi atas perbuatan tersebut.

Depok, 19 - Juni - 2026

Yang membuat pernyataan

(Ikhsan Fadillah)

NIM. 220742101

LEMBAR PENGESAHAN

Skripsi diajukan oleh:

Nama : Ikhsan Fadillah

NIM : 2207421013

Program Studi : T.Informatika dan Komputer / Teknik Multimedia dan Jaringan

Judul Skripsi : IMPLEMENTASI DEVSECOPS: MEKANISME HARD-BLOCKING PADA PIPELINE CI MENGGUNAKAN SAST SONARQUBE DAN JENKINS

Telah diuji oleh tim penguji dalam Sidang Skripsi pada hari Jumat,
Tanggal 19, Bulan Juni, Tahun 2026 dan dinyatakan
LULUS/~~TIDAK LULUS.~~

Disahkan oleh

Pembimbing I : Defiana Arnaldy, S.Tp., M.Si. ( )

Penguji I : Ayu Rosyida Zain, S.ST, M.T ( )

Penguji II : Fachroni Arbi Murad, S.Kom., M.Kom. ( )

Penguji III : Andika Riyadi Jasril, M.Pd.T. ( )

Mengetahui :

Jurusan Teknik Informatika dan Komputer

Ketua

Dr. Anita Hidayati, S.Kom., M.Kom.

NIP. 197908032003122003

KATA PENGANTAR

Puji syukur penulis panjatkan ke hadirat Tuhan Yang Maha Esa atas segala limpahan rahmat, karunia, dan kemudahan yang telah diberikan, sehingga penulisan skripsi ini dapat diselesaikan dengan baik. Laporan skripsi ini disusun sebagai salah satu syarat untuk memperoleh gelar Sarjana Terapan (Diploma 4) di Politeknik Negeri Jakarta.

Dalam kesempatan ini, penulis ingin menyampaikan penghargaan dan terima kasih yang sebesar-besarnya kepada berbagai pihak yang telah memberikan dukungan, bimbingan, serta motivasi selama proses penyusunan skripsi ini, di antaranya:

1. Kepada Bapak Defiana Arnaldy, S.Tp., M.Si. selaku dosen pembimbing yang telah memberikan arahan, bimbingan, serta saran-saran yang sangat berarti dalam setiap tahapan penulisan skripsi ini.
2. Kepada Bapak Masril dan Ibu Pepi Sri Indrawati selaku orang tua tercinta yang telah memberikan kepercayaan penuh, dukungan moral, dan fasilitas yang diperlukan selama studi dan penyusunan skripsi ini.
3. Kepada teman teman perjuangan skirpsi juga yang telah memberikan dukungan serta bantuan, baik secara moril maupun materil, sehingga penulis dapat terus bersemangat dalam menyelesaikan tugas akhir ini.

Penulis menyadari bahwa skripsi ini masih jauh dari sempurna, baik dari segi isi maupun penyajiannya. Oleh karena itu, kritik dan saran yang membangun sangat penulis harapkan demi perbaikan di masa mendatang.

Akhir kata, penulis berharap semoga skripsi ini dapat memberikan manfaat dan menjadi kontribusi yang berarti bagi pengembangan ilmu pengetahuan, khususnya di bidang yang relevan.

SURAT PERNYATAAN PERSETUJUAN PUBLIKASI SKRIPSI UNTUK KEPENTINGAN AKADEMIS

Sebagai sivitas akademik Politeknik Negeri Jakarta, saya bertanda tangan dibawah ini:

Nama	: Ikhsan Fadillah
NIM	: 2207421013
Jurusan/Program Studi	: Teknik Informatika dan Komputer/Teknik Multimedia dan Jaringan

Demi pengembangan ilmu pengetahuan, menyetujui untuk memberikan kepada Politeknik Negeri Jakarta Hak Bebas Royalti Non-Eksklusif atas karya ilmiah saya yang berjudul :

IMPLEMENTASI DEVSECOPS: MEKANISME HARD-BLOCKING PADA PIPELINE CI MENGGUNAKAN SAST SONARQUBE DAN JENKINS

Beserta perangkat yang ada (jika diperlukan). Dengan Hak Bebas Royalti Non-Eksklusif ini Politeknik Negeri Jakarta Berhak menyimpan, mengalihmediakan/formatkan, mengelola dalam bentuk pangkalan data (*database*), merawat, dan mempublikasikan skripsi saya tanpa meminta izin dari saya selama tetap mencantumkan nama saya sebagai penulis/pencipta dan sebagai pemilik Hak Cipta.

Demikian pernyatan ini saya buat dengan sebenarnya.

Depok, 19 - Juni - 2026

Yang membuat pernyataan

(Ikhsan Fadillah)

NIM. 2207421013

IMPLEMENTASI DEVSECOPS: MEKANISME HARD-BLOCKING PADA PIPELINE CI MENGGUNAKAN SAST SONARQUBE DAN JENKINS

Abstrak

Dalam ekosistem pengembangan perangkat lunak modern, penerapan metode DevOps sering menghadapkan tim pada dilema antara kecepatan rilis atau jaminan keamanan. Seringkali, aspek keamanan dikesampingkan demi mengejar waktu rilis *(time-to-market),* dan baru diperiksa setelah aplikasi selesai dibangun. Akibatnya, penanganan celah keamanan menjadi reaktif dan memakan biaya tinggi. Penelitian ini bertujuan mengatasi masalah tersebut dengan menerapkan DevSecOps, yaitu integrasi keamanan sejak dini (shift-left). Infrastruktur sistem diimplementasikan pada lingkungan server virtual terisolasi (*isolated virtual environment*) yang menjalankan kontainerisasi Docker untuk menjamin integritas pengujian tanpa intervensi dari sistem operasi utama. Jenkins digunakan sebagai orkestrator otomatisasi, sementara SonarQube berfungsi sebagai auditor kualitas kode dengan metode *Static Application Security Testing* (SAST). Studi kasus dilakukan pada aplikasi web berbasis Python yang memiliki celah keamanan kategori kesalahan konfigurasi *(Security Misconfiguration)*, seperti *Hardcoded Credentials*, penggunaan algoritma kriptografi usang (MD5), serta penggunaan protokol komunikasi yang tidak terenkripsi *(Clear-text Communication)*. Penelitian ini berfokus pada penerapan mekanisme "Gerbang Keamanan" (*Quality Gate*) dengan fitur pemblokiran otomatis (*Hard-Blocking*). Hasil pengujian membuktikan bahwa sistem secara konsisten mampu memblokir proses *build* dan integrasi kode pada pipeline CI secara otomatis apabila terdeteksi adanya pengaturan keamanan yang berbahaya. Arsitektur DevSecOps yang dibangun terbukti efektif menjamin keamanan aplikasi sebelum proses integrasi dinyatakan berhasil, sekaligus memastikan bahwa keamanan ini tidak memperlambat proses pengembangan.

Kata Kunci: DevSecOps, SAST, *Quality Gate*, *Hard-Blocking*, *Security Misconfiguration*.

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR LAMPIRAN

BAB I
PENDAHULUAN

1.1 Latar Belakang Masalah

Dalam era transformasi digital, kecepatan pengembangan perangkat lunak telah menjadi salah satu faktor utama yang menentukan daya saing organisasi teknologi. Metodologi DevOps diadopsi secara luas untuk mempercepat siklus pengembangan dan perilisan aplikasi melalui praktik *Continuous Integration* (CI) (Kim et al., 2021). Melalui pendekatan ini, tim pengembang dapat melakukan integrasi kode dan pengujian otomatis secara berulang dalam waktu yang lebih singkat. Namun, percepatan tersebut sering kali menempatkan aspek keamanan sebagai prioritas sekunder, karena fokus utama organisasi cenderung diarahkan pada kecepatan rilis dan *time-to-market* (Pranav et al., 2025). Akibatnya, pendekatan pengembangan tradisional yang memisahkan fase penulisan kode dan pengujian keamanan di akhir siklus masih menyisakan permasalahan serius berupa penumpukan kerentanan secara diam-diam dan peningkatan utang teknis (*technical debt*). Ketika pengujian keamanan baru dilakukan menjelang aplikasi dirilis, biaya, waktu, dan sumber daya yang dibutuhkan untuk memperbaiki celah keamanan menjadi jauh lebih besar serta dapat mengganggu ritme pengembangan secara keseluruhan.

Salah satu bentuk kelemahan keamanan yang paling sering muncul pada aplikasi modern adalah *Security Misconfiguration*. Berdasarkan OWASP Top 10 2021, kategori ini menempati peringkat kelima sebagai risiko keamanan aplikasi web yang paling kritis, dengan lebih dari 208.000 insiden kelalaian konfigurasi yang tercatat secara global (OWASP Foundation, 2021). *Security Misconfiguration* mencakup berbagai kesalahan konfigurasi yang pada awalnya tampak sepele, tetapi dapat berakibat fatal terhadap kerahasiaan, integritas, dan ketersediaan sistem. Contohnya adalah penggunaan kredensial atau kunci rahasia yang ditulis langsung di dalam kode sumber (*hardcoded credentials*), penggunaan algoritma kriptografi usang seperti MD5, konfigurasi cookie tanpa atribut keamanan yang memadai, hingga penggunaan protokol komunikasi tidak terenkripsi (*clear-text*

communication) yang membiarkan data sensitif terekspos di jaringan (MITRE Corporation, 2025).

Ancaman tersebut bukan hanya bersifat teoritis, tetapi telah terbukti menimbulkan dampak nyata dalam skala global. Laporan State of Secrets Sprawl 2024 dari GitGuardian mencatat adanya 12,8 juta insiden kebocoran kredensial rahasia di repositori kode publik sepanjang tahun 2023, yang menunjukkan bahwa hardcoded secret masih menjadi permasalahan serius dalam praktik pengembangan perangkat lunak modern (GitGuardian, 2024). Di sisi lain, laporan Cost of a Data Breach 2024 dari IBM Security menyebutkan bahwa rata-rata kerugian akibat insiden kebocoran data telah mencapai 4,88 juta dolar Amerika Serikat per insiden, dengan kesalahan konfigurasi dan lemahnya perlindungan data menjadi salah satu faktor penyebab yang dominan (IBM Security, 2024). Data tersebut menegaskan bahwa celah keamanan akibat miskonfigurasi tidak lagi dapat dipandang sebagai kesalahan kecil, melainkan sebagai risiko nyata yang berpotensi menimbulkan kerugian finansial, reputasi, dan operasional yang sangat besar bagi organisasi.

Merespons tingginya ancaman tersebut, sejumlah penelitian terdahulu telah menginisiasi integrasi keamanan ke dalam alur CI. Penelitian oleh Shama dan Chandra (2021) menunjukkan bahwa Jenkins dapat digunakan untuk mengotomatisasi proses integrasi dan pemindaian keamanan pada *pipeline*, sedangkan penelitian Rahmawati dan Susetyo (2023) membuktikan bahwa SonarQube mampu digunakan untuk memonitor kualitas kode dan mengidentifikasi berbagai temuan pada aplikasi. Meskipun demikian, mayoritas implementasi pada penelitian terdahulu masih bersifat pasif, yaitu sebatas menampilkan hasil analisis dan laporan kerentanan pada dasbor tanpa disertai mekanisme pencegahan otomatis. Dalam kondisi tersebut, *pipeline* masih dapat melanjutkan proses *build*, pembuatan kontainer, bahkan *deployment* meskipun mesin pemindai telah mendeteksi adanya celah keamanan. Pendekatan pasif seperti ini berisiko meloloskan aplikasi rentan ke lingkungan produksi serta memunculkan fenomena *alert fatigue*, yaitu kondisi ketika peringatan keamanan diabaikan karena tidak diikuti dengan tindakan teknis yang memaksa.

Berdasarkan permasalahan tersebut, penelitian ini merancang arsitektur DevSecOps yang bersifat proaktif dan preventif dengan menerapkan prinsip *Shift-Left Security*, yaitu menggeser proses pengujian keamanan ke tahap paling awal dalam siklus pengembangan perangkat lunak. Penelitian ini memfokuskan pengujian pada empat kerentanan *Security Misconfiguration*, yaitu *hardcoded credentials*, penggunaan algoritma kriptografi usang (MD5), konfigurasi *cookie* tanpa atribut *Secure*, dan komunikasi HTTP (*clear-text communication*). Inovasi utama yang ditawarkan adalah implementasi *Quality Gate* pada SonarQube dengan spesifikasi *zero tolerance* melalui mekanisme *Hard-Blocking*. Mekanisme ini tidak hanya berfungsi menampilkan laporan hasil pemindaian, tetapi juga bertindak secara deterministik untuk memutus dan menggagalkan *pipeline* CI secara otomatis ketika terdeteksi salah satu dari empat kerentanan tersebut pada *source code*. Dengan demikian, kode yang mengandung celah keamanan kritis dapat dicegah sejak tahap integrasi dan dipastikan tidak pernah mencapai tahap pembuatan kontainer maupun rilis akhir ke lingkungan produksi. Pendekatan ini diharapkan mampu memberikan kontribusi nyata dalam membangun *pipeline* DevSecOps yang tidak hanya cepat, tetapi juga memiliki kontrol keamanan yang ketat dan terukur.

1.2 Perumusan Masalah

Berdasarkan latar belakang yang telah diuraikan, maka perumusan masalah dalam penelitian ini adalah

1. Bagaimana merancang arsitektur *pipeline Continuous Integration* (CI) yang mengintegrasikan Jenkins dan SonarQube dalam lingkungan *server* virtual yang terisolasi?
2. Bagaimana efektivitas *ruleset* SAST SonarQube dalam mendeteksi berbagai variasi kerentanan *Security Misconfiguration* pada aplikasi berbasis Python?
3. Bagaimana penerapan mekanisme *Quality Gate* dengan fitur *hard-blocking* untuk mencegah proses *build* dinyatakan berhasil ketika ditemukan kerentanan keamanan kode yang memiliki kerentanan secara otomatis?

1.3 Batasan Masalah

Agar pembahasan lebih terarah dan tidak terlalu luas, penulis menetapkan batasan masalah sebagai berikut:

1. **Cakupan pengujian keamanan dibatasi pada metode SAST menggunakan SonarQube.**

   Penelitian ini hanya memanfaatkan pendekatan *Static Application Security Testing* (SAST) untuk mendeteksi kerentanan pada *source code*. Dengan demikian, penelitian belum mencakup metode pengujian keamanan lain seperti *Dynamic Application Security Testing* (DAST), *Software Composition Analysis* (SCA), maupun pengujian keamanan saat aplikasi sedang berjalan (*runtime security monitoring*). Akibatnya, hasil penelitian ini lebih merepresentasikan kemampuan mekanisme *hard-blocking* dalam merespons temuan kerentanan statis, bukan keseluruhan spektrum keamanan aplikasi.

2. **Objek kerentanan yang diuji dibatasi pada kategori *Security Misconfiguration* tertentu.**

   Penelitian ini hanya berfokus pada kerentanan *Security Misconfiguration* yang dipetakan ke beberapa skenario spesifik, yaitu *hardcoded credentials* (CWE-798), *weak cryptography* (CWE-327), *cookie misconfiguration* (CWE-614), *clear-text communication* (CWE-319), dan *debug mode aktif* (CWE-489) apabila skenario tersebut diikutsertakan secara penuh dalam pengujian. Oleh karena itu, hasil penelitian tidak dapat digeneralisasi secara langsung untuk seluruh jenis kerentanan aplikasi web seperti *injection*, *broken access control*, atau *cross-site scripting*.

3. **Lingkungan implementasi masih bersifat laboratorium dan belum merepresentasikan infrastruktur produksi berskala besar.** Arsitektur DevSecOps pada penelitian ini dibangun di lingkungan virtualisasi terisolasi menggunakan Oracle VM VirtualBox, Ubuntu Server, Docker, Jenkins, dan SonarQube Community Edition. Walaupun

pendekatan ini efektif untuk validasi konsep dan pengujian terkendali, kondisi tersebut belum sepenuhnya menggambarkan kompleksitas infrastruktur produksi nyata, misalnya penggunaan *cluster orchestration*, *distributed runners*, *multi-branch pipeline*, atau integrasi dengan sistem organisasi skala besar

4. **Aplikasi target utama dibatasi pada ekosistem Python berbasis Flask.** Pengujian terkontrol pada penelitian ini dilakukan menggunakan aplikasi dan skrip Python berbasis Flask. Konsekuensinya, efektivitas *ruleset* SonarQube serta perilaku *Quality Gate* yang diamati dalam penelitian ini paling relevan untuk konteks aplikasi Python, dan belum tentu identik jika diterapkan pada bahasa pemrograman, *framework*, atau pola arsitektur aplikasi lain.

5. **Pengukuran performa difokuskan pada waktu eksekusi *pipeline* dan konsistensi pemblokiran, belum sampai pada dampak operasional jangka panjang.**

   Evaluasi penelitian ini berfokus pada kemampuan sistem dalam mendeteksi kerentanan, memicu *hard-blocking*, menjaga determinisme hasil, dan mengukur durasi eksekusi *pipeline*. Penelitian ini belum mengevaluasi dampak implementasi DevSecOps dalam jangka panjang, seperti penurunan jumlah kerentanan pada siklus pengembangan berikutnya, perubahan perilaku pengembang terhadap keamanan kode, atau pengaruhnya terhadap produktivitas tim pada lingkungan kerja riil.

6. **Mekanisme yang dibangun berhenti pada tahap deteksi dan pencegahan, belum mencakup perbaikan otomatis.** Sistem yang dikembangkan dalam penelitian ini dirancang untuk mendeteksi kerentanan dan menghentikan *pipeline* secara otomatis ketika ditemukan pelanggaran *Quality Gate*. Namun, penelitian belum mengimplementasikan fitur *auto-remediation* atau pemberian saran perbaikan kode secara otomatis. Dengan demikian, proses perbaikan tetap harus dilakukan secara manual oleh pengembang setelah *pipeline* diblokir.

1.4 Tujuan dan Manfaat

1.4.1 Tujuan Penelitian

Tujuan yang ingin dicapai dari penelitian ini adalah:

1. Membangun purwarupa infrastruktur DevSecOps otomatis dalam lingkungan virtualisasi terisolasi menggunakan integrasi Jenkins dan SobarQube pada Docker.
2. Mengukur efektivitas metode SAST SonarQube dalam mendeteksi kerentanan spesifik *Security Misconfiguration* pada *source code* aplikasi Python sebelum aplikasi dirilis.
3. Mengimplementasikan dan memvalidasi aturan *Quality Gate* yang memutus proses *build* secara otomatis (*Hard-Blocking*) saat terdeteksi konfigurasi keamanan berisiko tinggi.

1.4.2 Manfaat Penelitian

Manfaat yang diharapkan dari penelitian ini adalah:

1. Bagi Akademisi: Memberikan referensi teknis mengenai implementasi arsitektur DevSecOps berbasis kontainer dan mekanisme *Quality Gate*.
2. Bagi Pengembang: Mempercepat *feedback loop* terkait keamanan kode sehingga perbaikan dapat dilakukan lebih dini (*early detection*) tanpa menunggu fase pengujian akhir.

1.5 Sistematika Penulisan

Untuk memberikan gambaran menyeluruh mengenai proposal skripsi ini, sistematika penulisan disusun sebagai berikut:

1. **BAB I PENDAHULUAN** Menguraikan latar belakang masalah, perumusan masalah, batasan masalah, tujuan dan manfaat penelitian, serta sistematika penulisan.
2. **BAB II TINJAUAN PUSTAKA** Membahas landasan teori penunjang seperti *DevSecOps*, *CI pipeline*, *SAST*, *OWASP Top 10*, *CWE*, perangkat lunak yang digunakan (*Jenkins*, *SonarQube*, *Docker*), virtualisasi terisolasi,

flaky tests, metrik *DORA*, serta tinjauan terhadap penelitian terdahulu yang relevan.

3. **BAB III METODE PENELITIAN** Menjelaskan rancangan penelitian eksperimental, langkah-langkah sistematis tahapan penelitian, tahap uji stres dan validasi eksternal, serta teknik pengumpulan data.
4. **BAB IV HASIL DAN PEMBAHASAN** Menguraikan analisis kebutuhan sistem, perancangan dan implementasi arsitektur *pipeline DevSecOps*, pengujian skenario celah keamanan (*Security Misconfiguration*), analisis data dan evaluasi respons pemblokiran paksa (*Hard-Blocking*), serta uji validasi keamanan skala besar.
5. **BAB V PENUTUP** Berisi kesimpulan akhir yang menjawab seluruh rumusan masalah penelitian, serta saran akademis maupun teknis untuk pengembangan arsitektur sistem di masa mendatang.

BAB V

PENUTUP

5.1 Kesimpulan

Berdasarkan hasil penelitian dan pengujian yang telah dilakukan terhadap implementasi mekanisme *Hard-Blocking* pada *pipeline* CI, dapat disimpulkan beberapa hal sebagai berikut:

1. Arsitektur *Pipeline* DevSecOps: Arsitektur *pipeline* DevSecOps berhasil dirancang dan diimplementasikan dengan mengintegrasikan Jenkins sebagai orkestrator dan SonarQube sebagai pemindai keamanan dalam lingkungan server virtual terisolasi menggunakan Docker. Konfigurasi ini terbukti mampu mengisolasi seluruh proses pengujian sehingga tidak memberikan dampak teknis pada sistem operasi utama dan menjamin lingkungan *sandbox* yang aman.

2. Efektivitas *Ruleset* SAST SonarQube: *Ruleset* SAST pada SonarQube memiliki efektivitas yang tinggi dalam mendeteksi berbagai variasi kerentanan *Security Misconfiguration* (OWASP A05:2021) pada aplikasi berbasis Python. Sistem berhasil secara konsisten mengidentifikasi lima kategori celah keamanan kritis, yaitu *Hardcoded Credentials* (CWE-798), *Weak Cryptography* (CWE-327), *Cookie Misconfiguration* (CWE-614), *Clear-text Communication* (CWE-319), dan *Leftover Debug Code* (CWE-489).

3. Mekanisme *Quality Gate* dan *Hard-Blocking*: Penerapan mekanisme *Quality Gate* dengan fitur *Hard-Blocking* terbukti sangat efektif dan deterministik dalam mencegah *deployment* kode yang tidak aman. Seluruh pengujian pada berbagai repositori target menunjukkan bahwa sistem secara otomatis menghentikan proses *build* (*FAILED*) setiap kali terdeteksi kerentanan yang melanggar ambang batas keamanan yang ditetapkan.

5.2 Saran

Untuk pengembangan sistem di masa mendatang, penulis menyarankan beberapa poin sebagai berikut:

1. Integrasi *Software Composition Analysis* (SCA): Mengingat penelitian ini hanya berfokus pada analisis kode statis (SAST), disarankan untuk menambahkan instrumen SCA agar sistem mampu mendeteksi kerentanan pada *library* atau dependensi pihak ketiga (*third-party dependencies*) yang sering kali menjadi pintu masuk serangan.

2. Otomatisasi Notifikasi *Real-Time*: Perlu dilakukan integrasi sistem notifikasi otomatis ke *platform* komunikasi seperti Discord, Slack, atau Email. Hal ini bertujuan agar pengembang mendapatkan umpan balik *Hard-Blocking* secara instan tanpa harus memantau dasbor secara manual.

3. Pengembangan Fitur *Auto-Remediation*: Mengingat penelitian ini terbatas pada tahap deteksi dan pencegahan, pengembangan sistem ke depan dapat diarahkan untuk mengimplementasikan fitur perbaikan kode otomatis (*auto-remediation*) guna mempercepat siklus perbaikan celah keamanan bagi tim pengembang.

4. Pengujian DAST: Penambahan metode *Dynamic Application Security Testing* (DAST) disarankan untuk menguji keamanan aplikasi pada saat berjalan (*runtime*), sehingga memberikan lapisan pertahanan yang lebih komprehensif terhadap ancaman keamanan web.

DAFTAR PUSTAKA

Alshammari, A., Morris, C., Hilton, M. dan Bell, J. 2021. "FlakeFlagger: Predicting Flakiness Without Rerunning Tests". *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, Madrid, Spain, 1572-1584. https://ieeexplore.ieee.org/document/9402098. [14 April 2026]

Bhimireddy, B.R., Nimmagadda, A., Kurapati, H., Gogula, L.R., Chintala, R.R. dan Jadala, V.C. 2023. "Web Security and Web Application Security: Attacks and Prevention". *2023 8th International Conference on Computer and Communication Systems (ICCCS)*, Guangzhou, China, 628-632. https://ieeexplore.ieee.org/document/10112741. [21 Januari 2026]

Fadilah, N., Hartono, R. dan Anwar, D.S. 2025. "Pengembangan Prototype Arsitektur CI/CD Pada Sistem Informasi Manajemen Terintegrasi Universitas Perjuangan Menggunakan Jenkins". *JATI (Jurnal Mahasiswa Teknik Informatika)*, 9 (4), 6873-6881. https://www.ejournal.itn.ac.id/index.php/jati/article/view/13837. [20 Januari 2026]

Google Cloud. 2024. *DORA Metrics: Understanding Software Delivery Performance*. https://cloud.google.com/devops/state-of-devops/. [28 April 2026]

GitGuardian. 2024. *The State of Secrets Sprawl 2024: A Year in Review*. https://www.gitguardian.com/state-of-secrets-sprawl-report-2024. [22 Juni 2026]

IBM Security. 2024. *Cost of a Data Breach Report 2024*. https://www.ibm.com/reports/data-breach. [22 Juni 2026]

Kim, G., Humble, J., Debois, P., Willis, J. dan Forsgren, N. 2021. *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations* (2nd ed.). Portland: IT Revolution Press.

https://itrevolution.com/product/the-devops-handbook-2nd-edition/. [21 Januari 2026]

Mavani, C., Mistry, H., Goswami, A. dan Patel, R. 2024. "Real-Time Static and Dynamic Application Security in Production Environments". *International Journal for Multidisciplinary Research (IJFMR)*, 6 (4). https://www.ijfmr.com/papers/2024/4/51728.pdf. [20 Januari 2026]

MITRE Corporation. 2025. *Common Weakness Enumeration (CWE) Version 4.15*. https://cwe.mitre.org/. [14 April 2026]

OWASP Foundation. 2021. *OWASP Top 10:2021 The Critical Web Application Security Risks*. https://owasp.org/Top10/. [20 Januari 2026]

Portnoy, M. 2023. *Virtualization Essentials* (3rd ed.). Indianapolis: Sybex (Wiley).

Pranav, M., Madhesh, I., Lenin, J. dan Sasikumar, R. 2025. "Advances in DevSecOps and the Future of Cybersecurity using Automation". *EAI Endorsed Transactions on Pervasive Health and Technology*, 11, 1-12. https://eudl.eu/doi/10.4108/eai.28-4-2025.2357955. [20 Januari 2026]

Rahmawati, A.F. dan Susetyo, Y.A. 2023. "Analisis Quality Code Menggunakan SonarQube dalam Suatu Aplikasi Berbasis Laravel". *IT-Explore: Jurnal Penerapan Teknologi Informasi dan Komunikasi*, 2 (2), 99-103. https://ejournal.uksw.edu/itexplore/article/view/8609. [20 Januari 2026]

Shama, A.M. dan Chandra, D.W. 2021. "Implementasi Static Application Security Testing Menggunakan Jenkins CI/CD Berbasis Docker Container Pada PT. Emporia Digital Raya". *Jurnal Ilmiah Informatika (JIF)*, 9 (2), 95-99. https://www.researchgate.net/publication/356582739. [20 Januari 2026]

SonarSource. 2025. *SonarQube User Guide: Quality Gates and Security Metrics*. https://docs.sonarsource.com/sonarqube/latest/user-guide/quality-gates/. [7 Mei 2026

DAFTAR RIWAYAT HIDUP

DAFTAR RIWAYAT HIDUP PENULIS

Ikhsan Fadillah dilahirkan di Tangerang pada tanggal 5 oktober 2003. Penulis menyelesaikan pendidikan menengah kejuruan dan dinyatakan lulus dari SMKN 1 Kabupaten Tangerang pada tahun 2022. Setelah menyelesaikan pendidikan di tingkat sekolah menengah, penulis melanjutkan pendidikan ke jenjang pendidikan tinggi jenjang Sarjana Terapan (Diploma 4) dan tercatat sebagai mahasiswa Program Studi Teknik Multimedia dan Jaringan, Jurusan Teknik Informatika dan Komputer, Politeknik Negeri Jakarta angkatan tahun 2022.

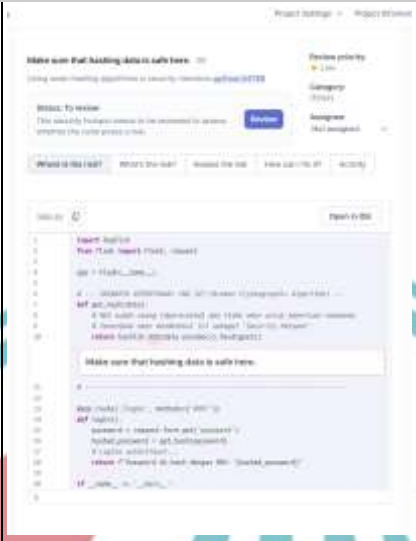
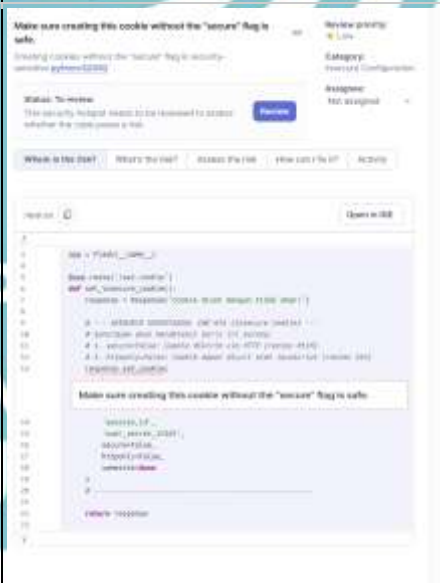
LAMPIRAN

LAMPIRAN 1

Katalog Bukti Visual Pengujian Kerentanan dan Pemblokiran *Pipeline*

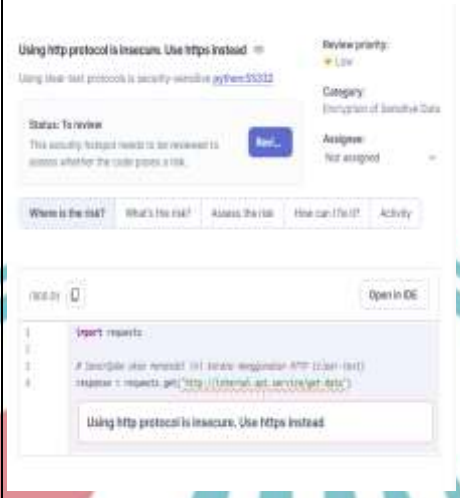
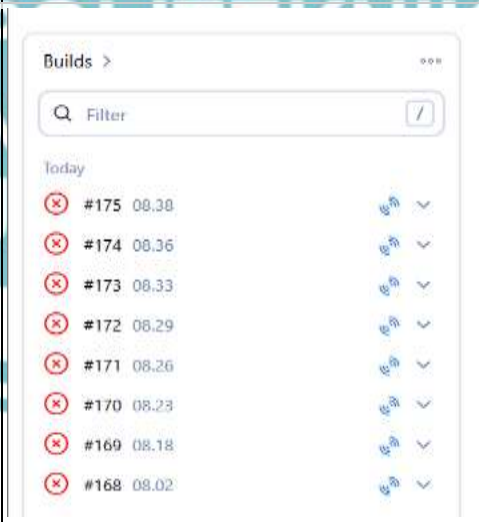
Tabel L.1 di bawah ini merupakan katalog referensi yang memetakan setiap skenario pengujian kerentanan keamanan siber dengan bukti visual yang dihasilkan oleh sistem. Katalog ini mencakup hasil tangkapan layar dari antarmuka SonarQube untuk validasi jenis kerentanan dan antarmuka Jenkins untuk validasi mekanisme pemblokiran paksa (*Hard-Blocking*).

No	Skenario Kerentanan (ID CWE)	Nama Berkas Sumber	Deskripsi Bukti Visual	Kode Lampiran
1	Baseline (Sistem Aman)	app.py	 Dasbor Jenkins menunjukkan status *Passed* dan *Quality Gate* hijau.	Lampiran A.1

2	Weak Cryptography (CWE-327)	app.py	 Deteksi penggunaan algoritma hashing MD5 pada Security Hotspot.	Lampiran A.3
3	Kerentanan Atribut Kuki (CWE-614)	app.py	 Pastikan pembuatan *cookie* ini tanpa flag "secure" aman.	Lampiran A.4

LAMPIRAN 1 (lanjutan)

Katalog Bukti Visual Pengujian Kerentanan dan Pemblokiran *Pipeline*

4	*Clear-text Communication* (CWE-319)	app.py	 Peringatan *"Using* http *protocol is insecure"* di SonarQube.	Lampiran A.5
5	Bukti *Hard-Blocking* (GUI)	Dasbor Jenkins	 Indikator lencana merah *(Failed)* pada antarmuka *pipeline.*	Lampiran A.5
6	Bukti riwayat eksekusi pemindaian *pipeline* secara berulang	Dasbor Jenkins		Lampiran A.6

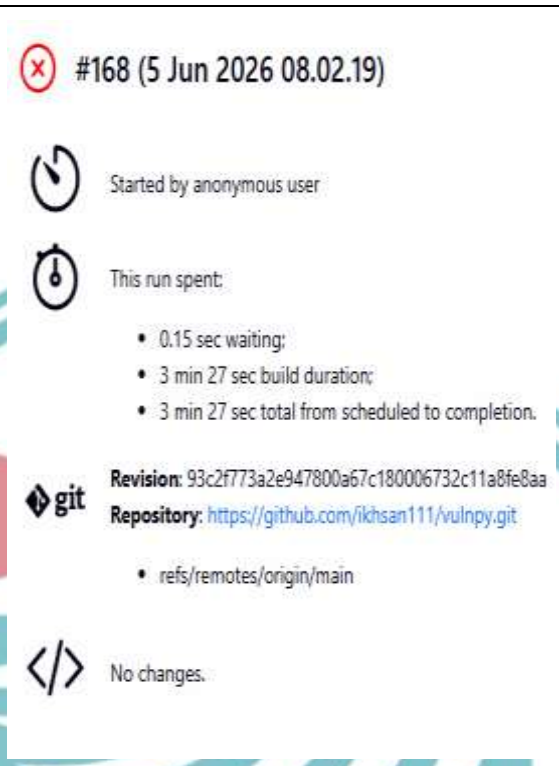
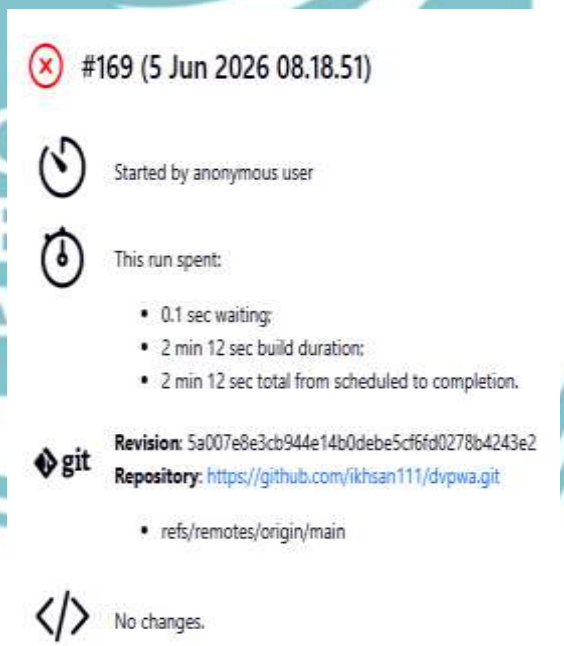
7	Keseluruhan log hasil mekanisme pemblokiran paksa (*Hard-Blocking*)		

LAMPIRAN 2

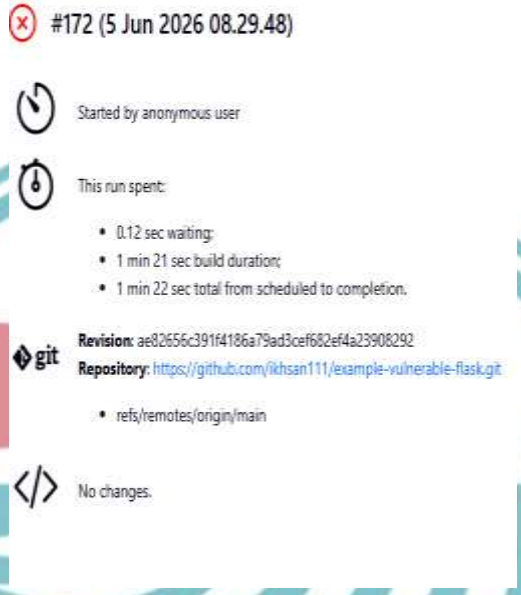
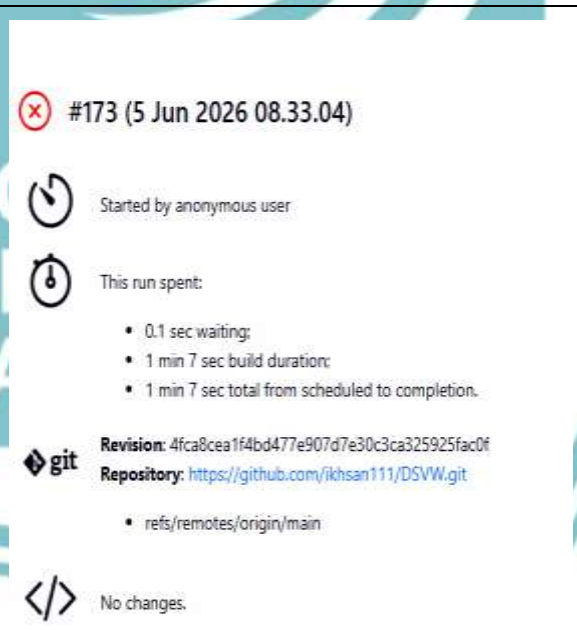
Rekapitulasi Hasil Uji Tambahan pada Tujuh Repositori

Tabel L.2 Rekapitulasi Hasil Uji Tambahan pada Tujuh Repositori

No	Nama Repositori	Status Jenkins	Bukti

1.	vulnpy (Contrast Security)	X *FAILED*	
2.	dvpwa (Damn Vulnerable Python)	X *FAILED*	

3.	vfapi (FastAPI Vulnerable)	X *FAILED*	
4.	Vulnerable-API	X *FAILED*	

5.	example-vulnerable-flask	X *FAILED*	
6.	DSVW (Damn Small Vulnerable Web)	X *FAILED*	

7.	Vulnerable-Flask-Appv-videveloper s	X *FAILED*