Analisis dan Mitigasi Keamanan Aplikasi E-Raport Menggunakan OWASP Top 10

Aisyah Putri Sabrina[1], Fajriyanto[2]
[1,2,3] Program Studi Sistem Informasi, Universitas Ibrahimy, Situbondo
Email: Aisyahputrisabrina31@gmail.com[1], fajri17riyan@gmail.com[2]

Abstrak

Aplikasi E-Raport menyimpan data akademik, identitas siswa, dan aktivitas penilaian yang bernilai sensitif sehingga membutuhkan pengujian keamanan sejak tahap pengembangan. Penelitian ini bertujuan menganalisis celah keamanan pada kode sumber aplikasi E-Raport berbasis Laravel dengan pendekatan OWASP Top 10 dan Static Application Security Testing (SAST), kemudian menyusun langkah mitigasi yang dapat diterapkan langsung oleh pengembang. Metode penelitian menggunakan eksperimen laboratorium dengan alur shift-left security, meliputi pemetaan aset kode, konfigurasi Snyk Security Analyzer dan SonarQube for IDE, pemindaian statis, klasifikasi temuan berdasarkan OWASP dan CWE, refaktorisasi kode, serta validasi ulang. Hasil penelitian menunjukkan tiga kelompok risiko utama, yaitu penggunaan kredensial statis pada model dan pengujian otomatis, kelemahan kriptografi akibat dependensi rentan, serta kurangnya pengendalian konfigurasi keamanan. Mitigasi dilakukan melalui pemindahan rahasia ke environment variable, hashing dinamis, pembaruan dependensi, dan validasi ulang hingga panel pemindai menunjukkan kondisi clean state. Penelitian ini menegaskan bahwa integrasi SAST di IDE efektif sebagai kontrol awal untuk menurunkan risiko keamanan aplikasi pendidikan sebelum kode masuk ke repositori produksi.

Kata Kunci: E-Raport, OWASP Top 10, SAST, Laravel, Shift-Left Security

Abstract

E-Raport applications process academic records, student identities, and assessment activities that require security controls from the earliest development stage. This study analyzes security weaknesses in the source code of a Laravel-based E-Raport application using the OWASP Top 10 perspective and Static Application Security Testing (SAST), then formulates practical mitigation steps for developers. The research applies a laboratory experiment based on the shift-left security workflow, consisting of code asset mapping, configuration of Snyk Security Analyzer and SonarQube for IDE, static scanning, classification of findings using OWASP and CWE, code refactoring, and rescanning for validation. The results reveal three major risk groups: hardcoded credentials in model and automated testing components, cryptographic weaknesses related to vulnerable dependencies, and insufficient security configuration control. Mitigation was performed by moving secrets into environment variables, applying dynamic hashing, upgrading dependencies, and validating the result until the scanner reported a clean state. The study confirms that IDE-integrated SAST is useful as an early control to reduce security risks in educational web applications before code is merged into production repositories.

Keywords: E-Raport, OWASP Top 10, SAST, Laravel, Shift-Left Security

PENDAHULUAN

Perkembangan teknologi informasi telah mendorong transformasi digital di berbagai sektor, termasuk sektor pendidikan. Salah satu bentuk implementasi transformasi tersebut adalah penggunaan aplikasi E-Raport sebagai media pengelolaan nilai, penyimpanan data akademik, dan penyajian laporan hasil belajar siswa secara elektronik.

Penggunaan aplikasi berbasis web memberikan berbagai keuntungan, seperti kemudahan akses, efisiensi pengolahan data, serta percepatan penyampaian informasi kepada guru, siswa, maupun pihak sekolah. Namun, di balik berbagai kemudahan tersebut, aplikasi berbasis web juga menghadapi tantangan yang besar dalam aspek keamanan informasi. Apabila aspek keamanan tidak diperhatikan sejak tahap pengembangan, maka aplikasi berpotensi mengalami berbagai serangan siber yang dapat menyebabkan kebocoran data, manipulasi informasi akademik, maupun gangguan terhadap layanan sistem.

Seiring meningkatnya kompleksitas aplikasi web, penerapan *secure coding* menjadi salah satu strategi utama untuk mencegah munculnya kerentanan keamanan sejak tahap pengembangan. Berbagai kerangka kerja seperti OWASP Top 10, NIST Cybersecurity Framework, dan Secure Software Development Lifecycle (SSDLC) direkomendasikan sebagai acuan dalam membangun perangkat lunak yang lebih aman(Kiashemshaki et al., n.d.).

Keamanan aplikasi web saat ini menjadi salah satu aspek yang sangat penting karena meningkatnya jumlah serangan siber yang menargetkan aplikasi berbasis internet. Data akademik yang tersimpan dalam aplikasi E-Raport termasuk kategori data sensitif karena memuat identitas pengguna, nilai siswa, data guru, serta informasi administrasi sekolah. Apabila data tersebut berhasil diakses oleh pihak yang tidak berwenang, maka dapat menimbulkan kerugian bagi institusi pendidikan maupun pengguna sistem. Oleh karena itu, proses pengembangan perangkat lunak tidak hanya berfokus pada fungsionalitas aplikasi, tetapi juga harus memperhatikan keamanan kode sumber (*secure coding*) dan keamanan komponen perangkat lunak yang digunakan.

Salah satu standar yang banyak digunakan dalam mengevaluasi keamanan aplikasi web adalah OWASP Top 10. Standar ini diterbitkan oleh Open Worldwide Application Security Project (OWASP) sebagai daftar sepuluh kategori kerentanan keamanan aplikasi web yang paling kritis berdasarkan tren serangan yang banyak ditemukan di dunia. OWASP Top 10 menjadi acuan bagi pengembang perangkat lunak, peneliti, maupun auditor keamanan dalam melakukan identifikasi, analisis, dan mitigasi terhadap berbagai kelemahan keamanan aplikasi. Kategori kerentanan tersebut meliputi *Broken Access Control*, *Cryptographic Failures*, *Injection*, *Insecure Design*, *Security Misconfiguration*, *Vulnerable and Outdated Components*, *Identification and Authentication Failures*, *Software and Data Integrity Failures*, *Security Logging and Monitoring Failures*, serta *Server-Side Request Forgery (SSRF)*.

Pendekatan Static Application Security Testing (SAST) merupakan salah satu metode yang efektif untuk mengidentifikasi kerentanan sejak tahap pengembangan perangkat lunak. Berbeda dengan metode pengujian yang dilakukan ketika aplikasi sedang berjalan, SAST melakukan analisis langsung terhadap kode sumber tanpa harus menjalankan aplikasi. Pendekatan ini memungkinkan pengembang menemukan potensi kerentanan lebih awal sehingga biaya perbaikan menjadi lebih rendah dibandingkan apabila kerentanan baru ditemukan pada tahap implementasi atau produksi. Berbagai perangkat lunak analisis statis telah dikembangkan untuk mendukung proses tersebut, di antaranya SonarQube dan Snyk Code. Selain itu, Snyk Open Source dapat digunakan untuk mengidentifikasi kerentanan yang berasal dari pustaka (*dependency*) pihak ketiga yang digunakan oleh aplikasi(Wagner et al., 2025).

Beberapa penelitian sebelumnya telah memanfaatkan OWASP maupun alat analisis keamanan dalam mengevaluasi aplikasi web. Aditya dkk. menganalisis keamanan website menggunakan OWASP ZAP dan menemukan beberapa kerentanan

yang memerlukan perbaikan untuk meningkatkan keamanan sistem. Sementara itu, Febriani dkk. melakukan analisis kerentanan pada sistem informasi akademik menggunakan OWASP ZAP dan menunjukkan bahwa evaluasi keamanan secara berkala mampu membantu organisasi dalam mengurangi risiko serangan siber. Penelitian lain juga menunjukkan bahwa penerapan praktik DevSecOps dan penggunaan alat analisis keamanan secara otomatis mampu meningkatkan kualitas keamanan perangkat lunak sejak tahap pengembangan.

Meskipun demikian, sebagian besar penelitian terdahulu lebih berfokus pada Dynamic Application Security Testing (DAST) menggunakan OWASP ZAP atau pengujian terhadap aplikasi yang sedang berjalan. Penelitian yang secara khusus mengombinasikan analisis kode sumber (SAST) menggunakan SonarQube dan Snyk Code, serta analisis kerentanan pustaka menggunakan Snyk Open Source pada aplikasi E-Raport berbasis Laravel masih relatif terbatas. Selain itu, penelitian sebelumnya umumnya berhenti pada tahap identifikasi kerentanan tanpa melakukan dokumentasi proses mitigasi secara menyeluruh hingga tahap verifikasi hasil perbaikan. Kondisi tersebut menunjukkan adanya peluang penelitian untuk mengkaji efektivitas proses identifikasi sekaligus mitigasi kerentanan keamanan berdasarkan kategori OWASP Top 10 pada aplikasi E-Raport.

Berdasarkan permasalahan tersebut, penelitian ini bertujuan untuk melakukan analisis terhadap kerentanan keamanan pada kode sumber aplikasi E-Raport menggunakan SonarQube, Snyk Code, dan Snyk Open Source. Setelah proses identifikasi dilakukan, penelitian ini menerapkan berbagai tindakan mitigasi sesuai rekomendasi keamanan yang relevan, seperti penghapusan *hardcoded password*, pengurangan duplikasi kode, pembaruan pustaka yang memiliki kerentanan, serta penerapan praktik *secure coding*. Efektivitas mitigasi kemudian dievaluasi kembali menggunakan alat analisis yang sama untuk mengetahui perubahan jumlah temuan kerentanan setelah proses perbaikan dilakukan.

Kontribusi penelitian ini terletak pada penyajian proses analisis keamanan kode sumber yang terintegrasi dengan tahapan mitigasi berdasarkan OWASP Top 10 pada aplikasi E-Raport berbasis Laravel. Hasil penelitian diharapkan dapat menjadi referensi bagi pengembang perangkat lunak pendidikan dalam meningkatkan keamanan aplikasi sejak tahap pengembangan serta memberikan gambaran penerapan SAST sebagai bagian dari proses pengembangan perangkat lunak yang lebih aman. Selain itu, penelitian ini juga dapat menjadi acuan bagi institusi pendidikan dalam melakukan evaluasi keamanan terhadap sistem informasi akademik yang digunakan sehingga risiko kebocoran data maupun serangan terhadap aplikasi dapat diminimalkan.

METODE

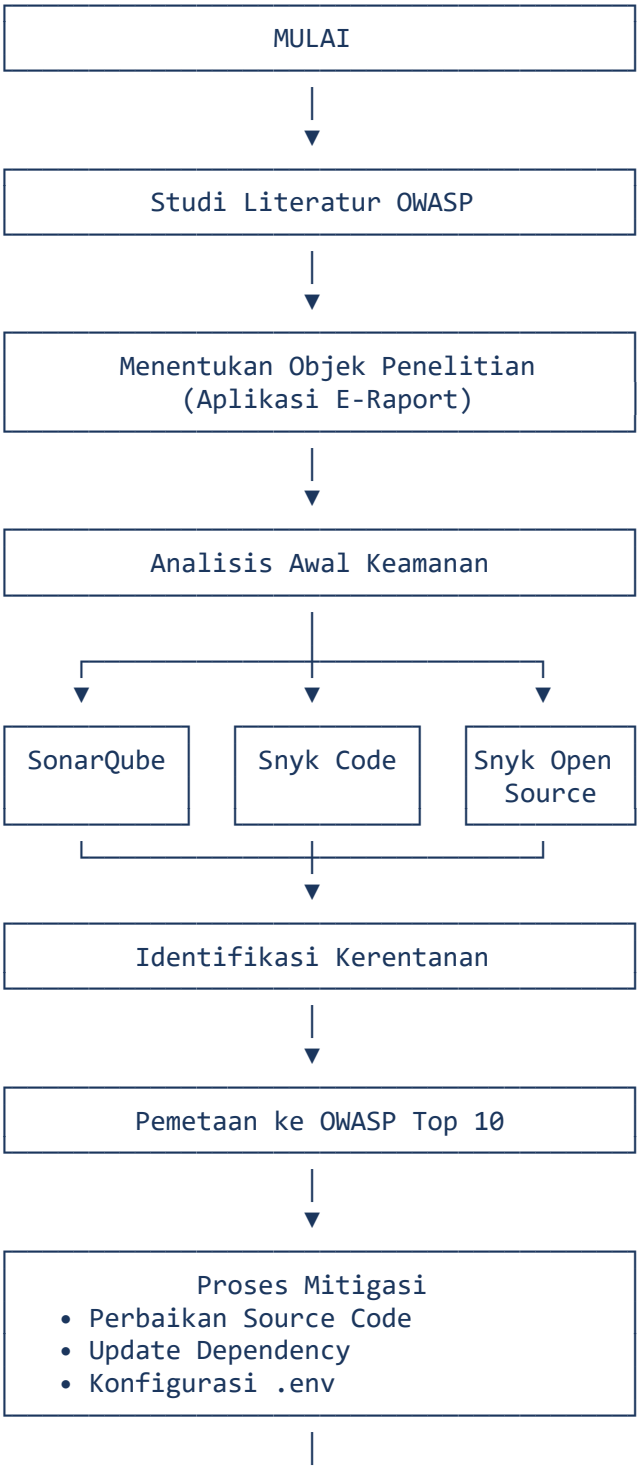
2.1 Objek Penelitian

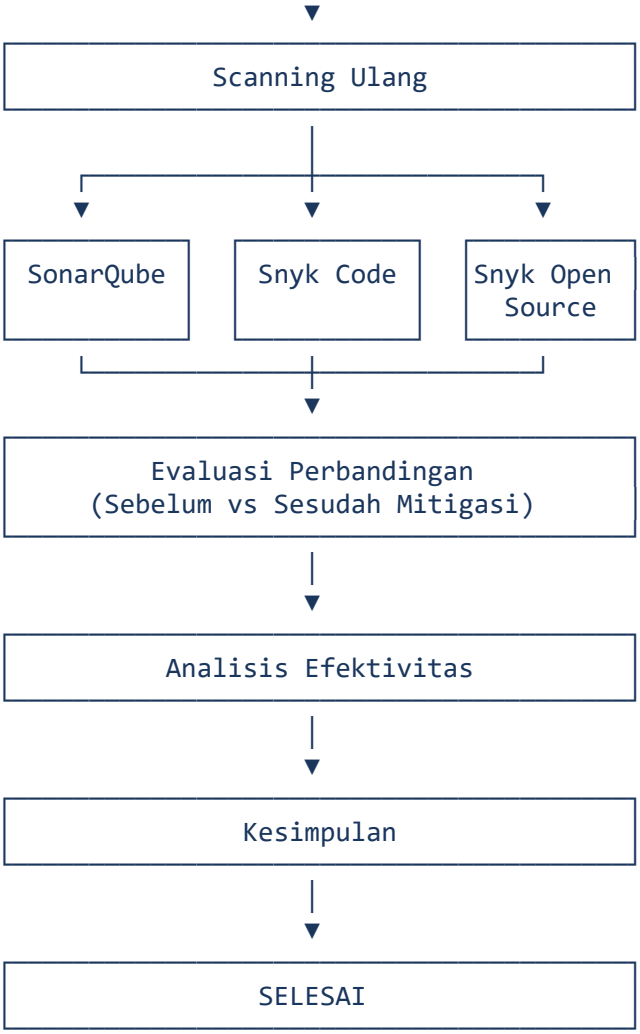
Objek penelitian ini adalah aplikasi E-Raport berbasis web yang digunakan untuk mengelola data akademik siswa pada tingkat sekolah dasar. Aplikasi dikembangkan menggunakan framework Laravel versi 12 dengan bahasa pemrograman PHP 8.2 dan sistem manajemen basis data MySQL. Fitur utama aplikasi meliputi autentikasi pengguna, pengelolaan data guru, data siswa, data kelas, mata pelajaran, jadwal pembelajaran, pengolahan nilai, serta pencetakan laporan hasil belajar siswa.

Penelitian difokuskan pada analisis keamanan kode sumber (*source code*) dan komponen perangkat lunak (*software dependency*) yang digunakan oleh aplikasi. Analisis dilakukan untuk mengidentifikasi potensi kerentanan keamanan yang termasuk dalam kategori OWASP Top 10 2021**,** kemudian dilakukan mitigasi sesuai rekomendasi keamanan sehingga kualitas keamanan aplikasi dapat ditingkatkan.

2.2 Tahapan Penelitian

Penelitian dilaksanakan melalui beberapa tahapan yang disusun secara sistematis. Tahapan dimulai dengan pengumpulan kode sumber aplikasi E-Raport, dilanjutkan dengan proses analisis keamanan menggunakan SonarQube, Snyk Code, dan Snyk Open Source. Selanjutnya, seluruh temuan kerentanan dipetakan ke dalam kategori OWASP Top 10 2021 sebagai dasar dalam menentukan strategi mitigasi. Setelah proses mitigasi selesai dilakukan, aplikasi dianalisis kembali menggunakan alat yang sama untuk mengevaluasi efektivitas perbaikan. Alur penelitian ditunjukkan pada **Gambar 1**.





Gambar 1 diagram alur

2.3 Alat Analisis Keamanan

Penelitian ini menggunakan tiga perangkat lunak analisis keamanan yang memiliki fungsi berbeda sehingga mampu memberikan hasil evaluasi yang saling melengkapi.

Tabel 1. Alat Analisis Keamanan yang Digunakan

Alat	Fungsi
SonarQube	Menganalisis kualitas kode sumber, *code smell*, *security hotspot*, serta potensi kerentanan keamanan pada aplikasi.
Snyk Code	Melakukan *Static Application Security Testing* (SAST) untuk mendeteksi kerentanan keamanan langsung pada kode sumber tanpa menjalankan aplikasi.
Snyk Open Source	Mengidentifikasi kerentanan keamanan yang berasal dari pustaka (*dependency*) pihak ketiga yang digunakan oleh aplikasi melalui Composer.

Ketiga alat tersebut digunakan secara berurutan untuk memperoleh hasil analisis yang komprehensif. SonarQube digunakan untuk mengevaluasi kualitas implementasi kode, sedangkan Snyk Code berfokus pada identifikasi kelemahan keamanan pada

kode sumber. Snyk Open Source digunakan untuk mendeteksi kerentanan pada pustaka eksternal sehingga proses mitigasi tidak hanya terbatas pada kode aplikasi, tetapi juga mencakup komponen perangkat lunak yang digunakan.

2.4 Prosedur Mitigasi dan Evaluasi

Tahapan mitigasi dilakukan berdasarkan hasil analisis yang diperoleh dari SonarQube, Snyk Code, dan Snyk Open Source. Seluruh temuan terlebih dahulu diklasifikasikan sesuai kategori OWASP Top 10 2021, kemudian dilakukan tindakan perbaikan sesuai karakteristik masing-masing kerentanan.

Mitigasi meliputi penghapusan kredensial yang ditulis secara langsung (*hardcoded credentials*), pengurangan duplikasi kode (*duplicate literals*), penghapusan kode yang tidak digunakan (*commented code*), pembaruan pustaka yang memiliki kerentanan melalui Composer, serta penerapan praktik *secure coding* pada bagian kode yang teridentifikasi berisiko.

Setelah seluruh proses mitigasi selesai, dilakukan analisis ulang menggunakan SonarQube dan Snyk untuk mengetahui perubahan jumlah temuan keamanan dibandingkan dengan kondisi sebelum mitigasi. Efektivitas mitigasi dievaluasi berdasarkan beberapa indikator, yaitu:

1. Penurunan jumlah *issue* yang ditemukan oleh SonarQube.
2. Penurunan jumlah kerentanan yang terdeteksi oleh Snyk Code.
3. Penurunan jumlah kerentanan pada *dependency* berdasarkan hasil Snyk Open Source.
4. Kesesuaian hasil mitigasi terhadap kategori OWASP Top 10 2021.
5. Keberhasilan aplikasi tetap berjalan dengan baik setelah proses mitigasi dilakukan.

HASIL DAN PEMBAHASAN

3.1 Hasil Analisis Keamanan Sebelum Mitigasi

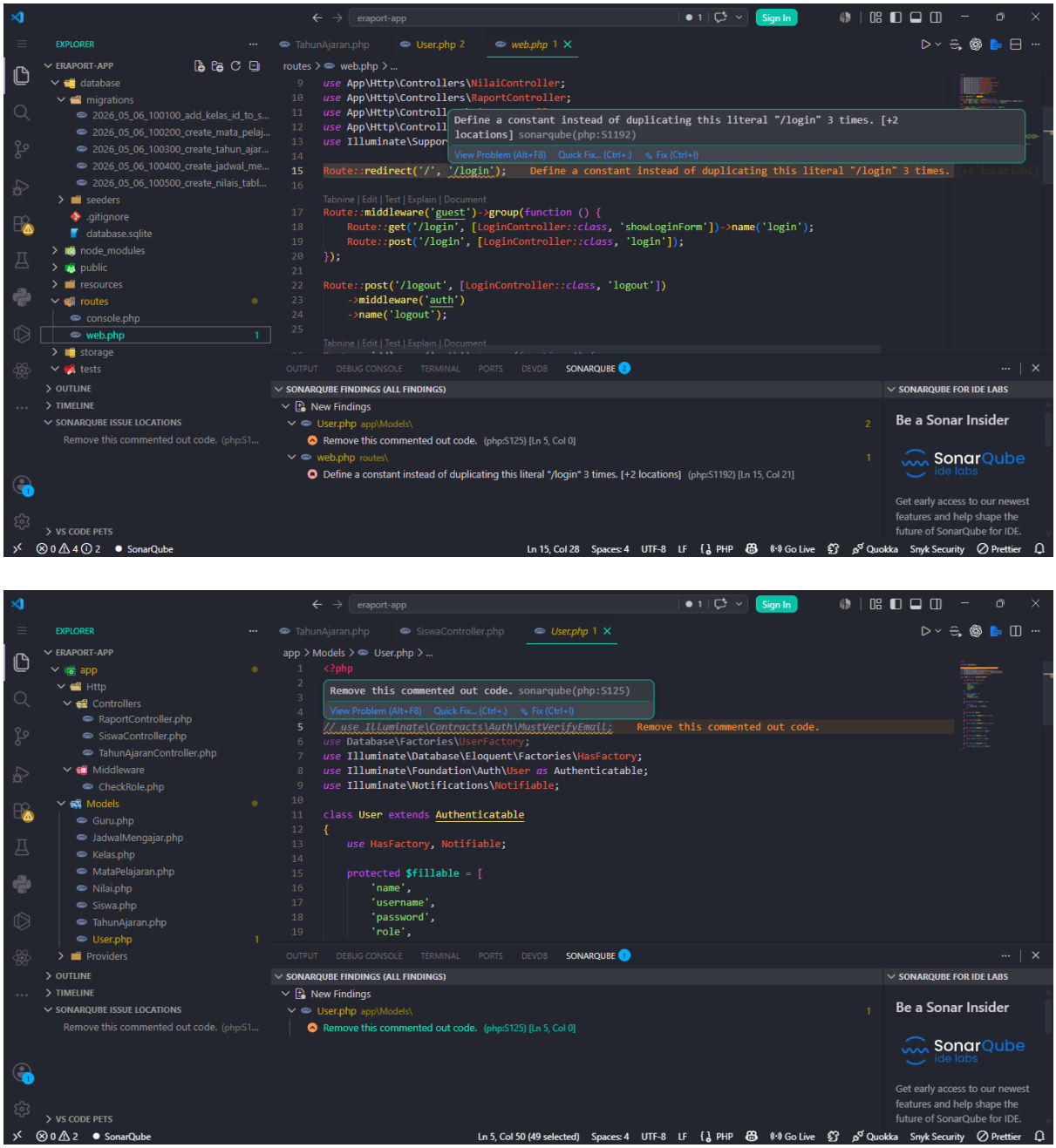
Tahap pertama penelitian dilakukan dengan melakukan analisis keamanan terhadap aplikasi E-Raport menggunakan tiga alat analisis, yaitu SonarQube, Snyk Code, dan Snyk Open Source. Penggunaan tiga alat tersebut bertujuan untuk memperoleh hasil analisis yang lebih komprehensif karena masing-masing memiliki fokus yang berbeda. SonarQube digunakan untuk mengevaluasi kualitas implementasi kode sumber dan mendeteksi *code smell* maupun *security hotspot*. Snyk

Code digunakan untuk melakukan *Static Application Security Testing* (SAST) sehingga mampu mengidentifikasi potensi kerentanan keamanan langsung dari kode sumber tanpa menjalankan aplikasi. Sementara itu, Snyk Open Source digunakan untuk menganalisis pustaka (*dependency*) yang digunakan aplikasi melalui Composer sehingga kerentanan yang berasal dari komponen pihak ketiga juga dapat diketahui.

Proses analisis dilakukan terhadap seluruh kode sumber aplikasi E-Raport berbasis Laravel versi 12 yang dikembangkan menggunakan PHP 8.2 dan basis data MySQL. Pemindaian dilakukan sebelum proses mitigasi sehingga seluruh hasil yang diperoleh menggambarkan kondisi awal keamanan aplikasi.

A. Hasil Analisis Menggunakan SonarQube

Hasil analisis menggunakan SonarQube menunjukkan bahwa masih terdapat beberapa temuan yang berkaitan dengan kualitas implementasi kode dan praktik *secure coding*. Berdasarkan hasil pemindaian, SonarQube mendeteksi penggunaan hardcoded password**,** duplicate literals, serta commented code yang masih terdapat pada beberapa bagian aplikasi.



Gambar 2. Hasil Analisis SonarQube Sebelum Mitigasi

Temuan *hardcoded password* menunjukkan bahwa informasi kredensial masih ditulis secara langsung di dalam kode sumber aplikasi. Praktik tersebut merupakan salah satu kelemahan keamanan yang cukup berbahaya karena informasi autentikasi dapat diketahui apabila kode sumber diakses oleh pihak yang tidak berwenang. Selain meningkatkan risiko kebocoran kredensial, penggunaan *hardcoded password* juga menyulitkan proses pemeliharaan aplikasi karena perubahan kata sandi harus dilakukan langsung pada kode program.

Dalam perspektif OWASP Top 10 2021, kelemahan ini berkaitan dengan kategori A07:2021 Identification and Authentication Failures. OWASP menjelaskan bahwa pengelolaan kredensial yang tidak tepat dapat menyebabkan kegagalan mekanisme autentikasi sehingga meningkatkan peluang terjadinya akses

tidak sah terhadap sistem. Oleh karena itu, informasi sensitif seperti nama pengguna, kata sandi, maupun *API key* seharusnya tidak ditulis secara langsung di dalam kode sumber, tetapi ditempatkan pada variabel lingkungan (*.env*) atau mekanisme pengelolaan rahasia (*secret management*)(Rahayuda et al., 2026).

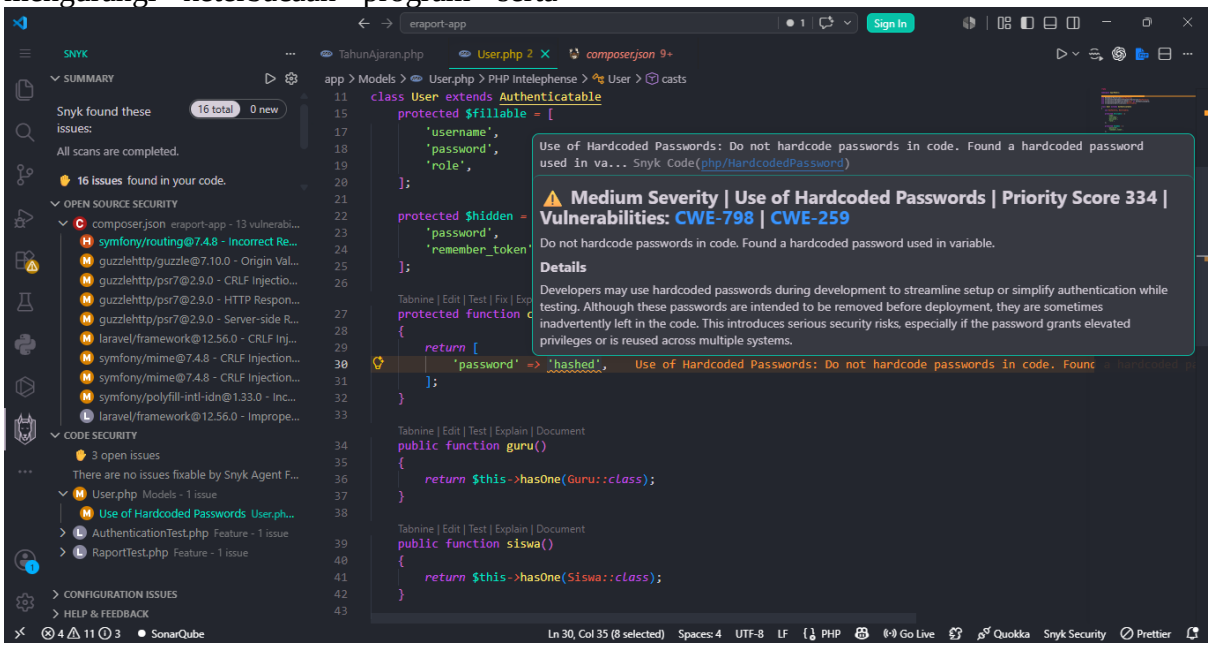
Selain *hardcoded password*, SonarQube juga mendeteksi keberadaan duplicate literals, yaitu penggunaan nilai yang sama secara berulang pada berbagai bagian kode. Meskipun temuan ini tidak secara langsung termasuk kategori kerentanan OWASP Top 10, kondisi tersebut dapat menurunkan *maintainability* aplikasi. Apabila terjadi perubahan nilai, pengembang harus memperbarui setiap bagian kode yang menggunakan literal tersebut sehingga meningkatkan kemungkinan munculnya inkonsistensi dan kesalahan implementasi.

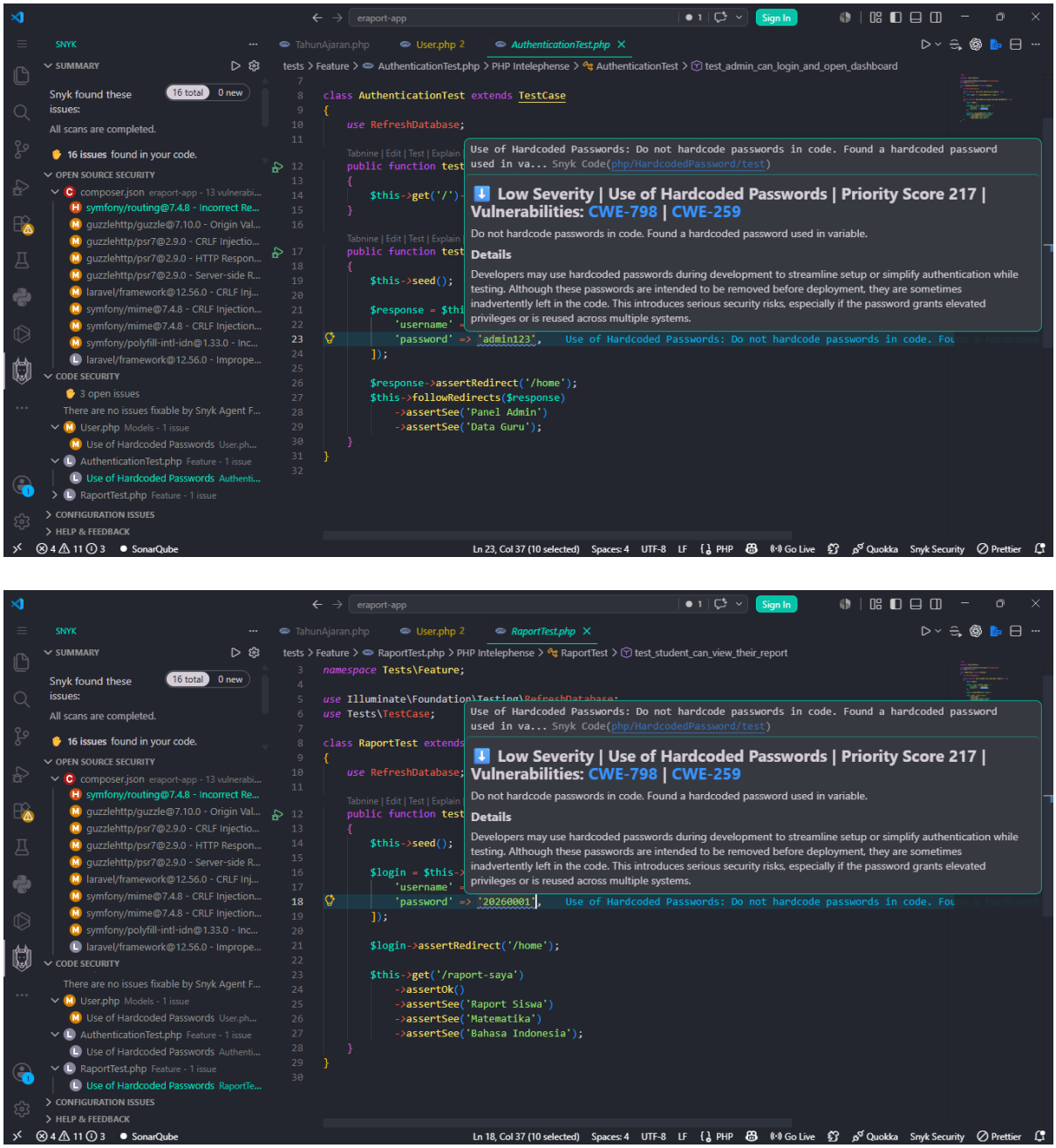
Temuan berikutnya adalah keberadaan commented code, yaitu potongan kode yang telah dinonaktifkan tetapi masih disimpan di dalam proyek. Keberadaan kode seperti ini dapat mengurangi keterbacaan program serta berpotensi membuka informasi mengenai implementasi lama yang seharusnya sudah tidak digunakan. Dalam praktik *secure coding*, kode yang tidak digunakan sebaiknya dihapus untuk menjaga kebersihan kode sumber dan mengurangi kemungkinan penyalahgunaan informasi oleh pihak yang tidak berwenang.

Secara keseluruhan, hasil analisis SonarQube menunjukkan bahwa aplikasi masih memerlukan perbaikan pada aspek kualitas implementasi kode sebelum dapat dikatakan memenuhi praktik pengembangan perangkat lunak yang aman.

B. Hasil Analisis Menggunakan Snyk Code

Analisis berikutnya dilakukan menggunakan Snyk Code. Berbeda dengan SonarQube yang lebih berorientasi pada kualitas kode, Snyk Code dirancang untuk mendeteksi kerentanan keamanan berdasarkan pendekatan *Static Application Security Testing* (SAST). Seluruh proses analisis dilakukan tanpa menjalankan aplikasi sehingga potensi kerentanan dapat diketahui sejak tahap pengembangan.





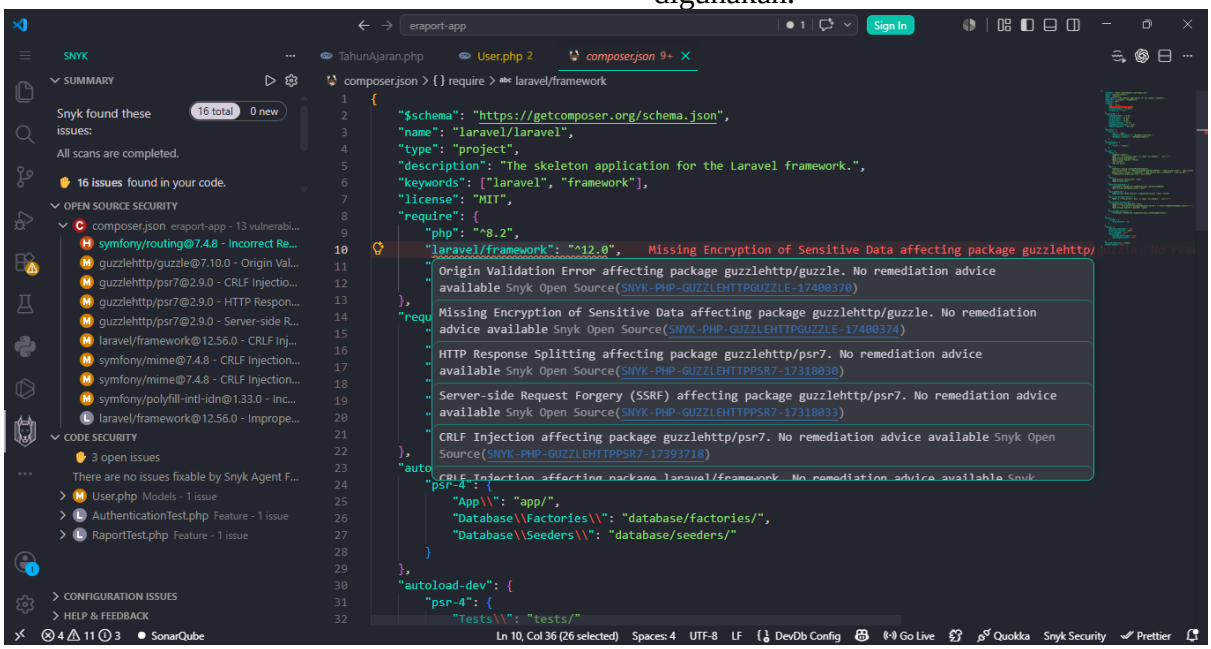
Gambar 3. Hasil Analisis Snyk Code Sebelum Mitigasi

Hasil pemindaian menunjukkan bahwa beberapa bagian kode masih memiliki praktik implementasi yang berpotensi menimbulkan risiko keamanan. Temuan tersebut berkaitan dengan pengelolaan kredensial dan penggunaan komponen tertentu yang belum memenuhi praktik *secure coding*. Walaupun jumlah temuan tidak sebanyak yang diperoleh dari analisis *dependency*, hasil ini menunjukkan bahwa evaluasi terhadap kode sumber tetap diperlukan untuk mencegah kerentanan berkembang menjadi celah keamanan yang dapat dieksploitasi.

Penggunaan Snyk Code pada penelitian ini memberikan manfaat karena alat tersebut mampu memberikan rekomendasi mitigasi secara langsung terhadap setiap temuan. Dengan demikian, pengembang dapat mengetahui lokasi kode yang bermasalah sekaligus memperoleh saran perbaikan yang sesuai dengan praktik keamanan modern.

C. Hasil Analisis Menggunakan Snyk Open Source

Selain menganalisis kode sumber, penelitian ini juga melakukan evaluasi terhadap pustaka eksternal yang digunakan aplikasi melalui Composer. Tahap ini penting karena sebagian besar aplikasi modern memanfaatkan banyak pustaka pihak ketiga sehingga keamanan aplikasi tidak hanya bergantung pada kualitas kode yang dikembangkan sendiri, tetapi juga pada keamanan komponen eksternal yang digunakan.



Gambar 4. Hasil Analisis Snyk Open Source Sebelum Mitigasi

Hasil analisis menunjukkan bahwa Snyk Open Source mendeteksi 16 kerentanan yang berasal dari beberapa *dependency*. Sebagian besar kerentanan disebabkan oleh penggunaan versi pustaka yang telah usang (*outdated*) sehingga telah memiliki catatan kerentanan pada basis data keamanan publik.

Menurut klasifikasi OWASP Top 10 2021, kondisi tersebut termasuk kategori A06:2021 Vulnerable and Outdated Components**.** Kerentanan pada komponen perangkat lunak merupakan salah satu penyebab utama terjadinya eksploitasi aplikasi karena penyerang dapat memanfaatkan kelemahan yang telah diketahui secara publik untuk memperoleh akses tidak sah atau menjalankan kode berbahaya.Hasil ini menunjukkan bahwa pembaruan *dependency* merupakan salah satu langkah mitigasi yang sangat penting.

Walaupun pengembang telah menulis kode dengan baik, penggunaan pustaka yang memiliki kerentanan tetap dapat menyebabkan aplikasi berada pada kondisi yang tidak aman. Oleh karena itu, proses pemeliharaan perangkat lunak harus mencakup pembaruan versi pustaka secara berkala sesuai rekomendasi pengembang resmi.

Untuk memberikan ringkasan hasil analisis awal, seluruh temuan disajikan pada Tabel 2.

Tabel 2. Ringkasan Temuan Sebelum Mitigasi

Alat Analisis	Temuan Utama	Kategori OWASP
SonarQube	Hardcoded Password	A07:2021 Identification and Authentication Failures
SonarQube	Duplicate Literals	Berpengaruh pada maintainability kode
SonarQube	Commented Code	Praktik secure coding

Snyk Code	Kerentanan implementasi kode	Bergantung pada jenis temuan
Snyk Open Source	16 dependency rentan	A06:2021 Vulnerable and Outdated Components

Berdasarkan hasil analisis awal tersebut, dapat disimpulkan bahwa aplikasi E-Raport masih memiliki beberapa kelemahan keamanan yang berasal dari implementasi kode maupun penggunaan komponen perangkat lunak pihak ketiga. Oleh karena itu, diperlukan proses mitigasi yang terstruktur untuk mengurangi risiko eksploitasi sekaligus meningkatkan kualitas keamanan aplikasi. Hasil identifikasi ini menjadi dasar dalam menentukan strategi perbaikan yang dibahas pada subbab berikutnya.

3.2 Proses Mitigasi Kerentanan

Setelah seluruh kerentanan berhasil diidentifikasi melalui proses analisis awal, tahap berikutnya adalah melakukan mitigasi terhadap setiap temuan berdasarkan tingkat risiko dan rekomendasi yang diberikan oleh SonarQube, Snyk Code, serta Snyk Open Source. Proses mitigasi dilakukan secara bertahap dengan tujuan mengurangi kemungkinan eksploitasi terhadap aplikasi E-Raport tanpa mengubah fungsionalitas utama sistem.

Strategi mitigasi yang diterapkan pada penelitian ini mengacu pada praktik *secure coding*, pedoman keamanan Laravel, serta rekomendasi OWASP Top 10 2021. Setiap kerentanan dianalisis terlebih dahulu untuk menentukan penyebab utama (*root cause*), kemudian dilakukan tindakan perbaikan yang sesuai. Setelah seluruh perbaikan selesai diterapkan, aplikasi diuji kembali untuk memastikan seluruh fitur tetap berjalan dengan baik dan tidak terjadi kesalahan akibat perubahan kode.

A. Mitigasi Hardcoded Password

Salah satu temuan utama pada tahap analisis awal adalah keberadaan hardcoded password, yaitu informasi autentikasi yang ditulis secara langsung di dalam kode sumber aplikasi. Praktik ini merupakan salah satu kelemahan yang sering ditemukan pada aplikasi berbasis web karena dapat menyebabkan kebocoran informasi sensitif apabila repositori kode diakses oleh pihak yang tidak berwenang.
Pada aplikasi E-Raport, beberapa konfigurasi masih menggunakan nilai autentikasi yang dituliskan secara langsung sehingga SonarQube mengidentifikasinya sebagai *security hotspot*. Kondisi tersebut meningkatkan risiko penyalahgunaan kredensial, terutama apabila kode sumber dipublikasikan pada repositori bersama atau digunakan oleh lebih dari satu pengembang.

Untuk mengatasi permasalahan tersebut, seluruh informasi sensitif dipindahkan ke dalam berkas konfigurasi lingkungan (*.env*) yang disediakan oleh framework Laravel. Selanjutnya aplikasi mengakses informasi tersebut melalui fungsi env() maupun konfigurasi pada berkas config. Pendekatan ini merupakan mekanisme yang direkomendasikan oleh Laravel karena memisahkan konfigurasi sensitif dari kode program sehingga lebih aman dan mudah dikelola.

Penerapan variabel lingkungan memberikan beberapa keuntungan. Pertama, informasi sensitif tidak lagi tersimpan pada repositori proyek sehingga mengurangi risiko kebocoran kredensial. Kedua, proses perubahan konfigurasi menjadi lebih mudah karena pengembang cukup memperbarui nilai pada berkas .env tanpa harus mengubah kode program. Ketiga, pendekatan ini mendukung penerapan berbagai lingkungan pengembangan (*development*, *testing*, dan *production*) dengan konfigurasi yang berbeda tanpa memerlukan perubahan kode sumber.

Dalam konteks OWASP Top 10 2021, mitigasi ini secara langsung mengurangi risiko pada kategori A07:2021 Identification and Authentication **Failures**,

khususnya yang berkaitan dengan pengelolaan kredensial yang tidak aman.

B. Mitigasi Duplicate Literals

Temuan berikutnya adalah keberadaan **duplicate literals**, yaitu penggunaan nilai yang sama secara berulang pada beberapa bagian kode. Walaupun tidak secara langsung dikategorikan sebagai kerentanan keamanan, kondisi ini memengaruhi kualitas perangkat lunak karena meningkatkan kompleksitas pemeliharaan (*maintainability*).

Nilai yang ditulis berulang kali berpotensi menimbulkan inkonsistensi apabila suatu saat diperlukan perubahan. Pengembang harus memperbarui setiap bagian kode yang menggunakan nilai tersebut sehingga peluang terjadinya kesalahan semakin besar. Selain itu, kode menjadi lebih sulit dipahami karena banyaknya literal yang tersebar pada berbagai file.

Untuk mengatasi kondisi tersebut, literal yang digunakan berulang kali diubah menjadi konstanta ataupun variabel sehingga hanya didefinisikan pada satu lokasi. Dengan pendekatan ini, setiap perubahan cukup dilakukan pada satu bagian sehingga konsistensi kode dapat dipertahankan.

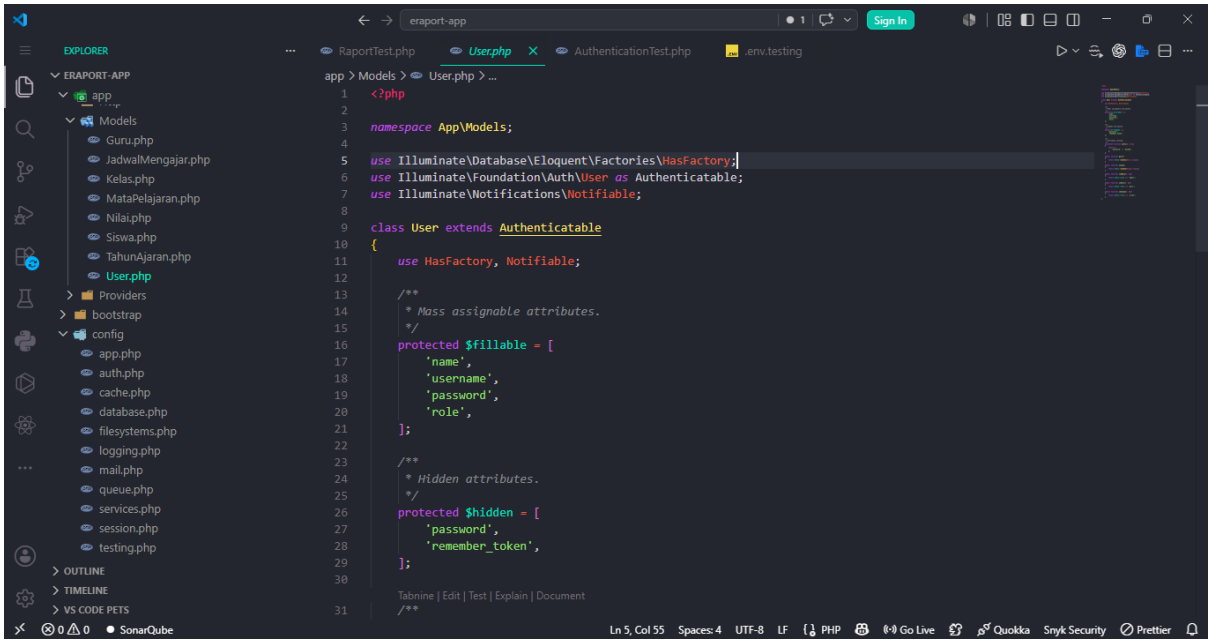
Selain meningkatkan kualitas kode, proses *refactoring* ini juga membuat struktur program menjadi lebih mudah dibaca, dipelihara, serta dikembangkan pada masa mendatang. Meskipun temuan ini tidak termasuk kategori OWASP Top 10 secara langsung, praktik tersebut merupakan bagian penting dari penerapan *secure coding* karena kualitas kode yang baik akan mengurangi kemungkinan munculnya kesalahan implementasi pada proses pengembangan berikutnya.

C. Penghapusan Commented Code

Hasil analisis SonarQube juga menunjukkan masih terdapat beberapa bagian kode yang telah dinonaktifkan (*commented code*) namun belum dihapus dari proyek. Keberadaan kode semacam ini sering dianggap tidak berbahaya, tetapi dalam praktik pengembangan perangkat lunak modern kondisi tersebut tidak direkomendasikan.

Kode yang tidak digunakan dapat membingungkan pengembang lain ketika melakukan pemeliharaan aplikasi. Selain itu, komentar yang berisi implementasi lama terkadang masih memuat informasi penting mengenai struktur sistem, logika bisnis, maupun konfigurasi tertentu yang seharusnya tidak lagi tersedia.

Sebagai langkah mitigasi, seluruh *commented code* yang sudah tidak digunakan dihapus dari proyek. Penghapusan dilakukan setelah dipastikan bahwa kode tersebut memang tidak lagi dibutuhkan oleh sistem.



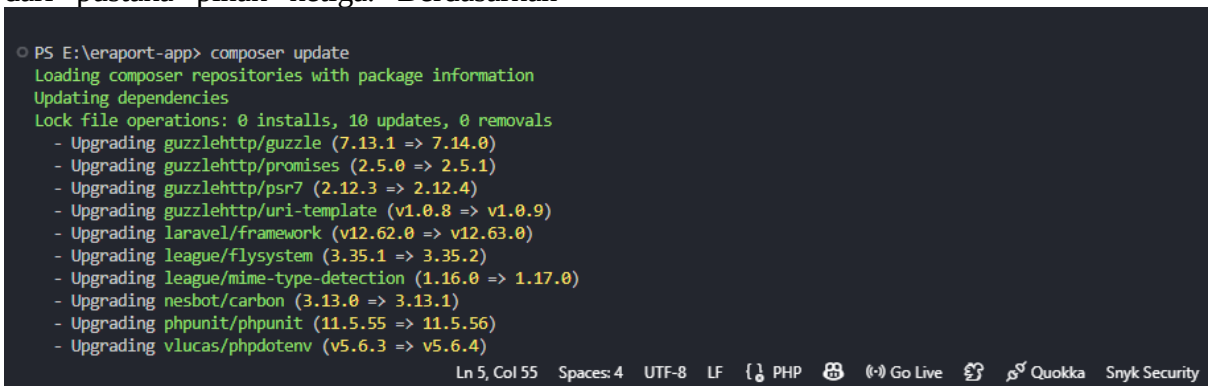
Gambar 5. Penghapusan Commented Code

Tindakan ini menghasilkan struktur kode yang lebih bersih, meningkatkan keterbacaan program, serta mempermudah proses audit keamanan pada tahap berikutnya. Walaupun tidak secara langsung diklasifikasikan sebagai kerentanan OWASP Top 10, praktik ini merupakan bagian dari prinsip pengembangan perangkat lunak yang aman dan mudah dipelihara.

D. Pembaruan Dependency Menggunakan Composer

Selain memperbaiki implementasi kode sumber, penelitian ini juga melakukan mitigasi terhadap kerentanan yang berasal dari pustaka pihak ketiga. Berdasarkan hasil analisis Snyk Open Source, ditemukan sejumlah *dependency* yang masih menggunakan versi lama sehingga memiliki kerentanan yang telah terdokumentasi pada basis data keamanan publik.

Langkah mitigasi dilakukan menggunakan **Composer** sebagai manajer *dependency* pada Laravel. Seluruh pustaka diperiksa menggunakan perintah composer audit untuk mengetahui daftar komponen yang memiliki kerentanan. Selanjutnya dilakukan pembaruan versi pustaka melalui perintah composer update sehingga aplikasi menggunakan versi terbaru yang telah memperoleh perbaikan keamanan dari pengembang.



Gambar 6. Proses Pembaruan Dependency Menggunakan Composer

Pembaruan dilakukan terhadap framework Laravel maupun beberapa pustaka Symfony yang sebelumnya masih menggunakan versi lama. Setelah proses pembaruan selesai, dilakukan kembali pemeriksaan menggunakan composer audit untuk memastikan tidak terdapat lagi *dependency* yang memiliki kerentanan keamanan.

Mitigasi terhadap *dependency* memiliki peran yang sangat penting karena sebagian besar aplikasi modern sangat bergantung pada pustaka eksternal. Walaupun kode sumber aplikasi telah ditulis dengan baik, penggunaan pustaka yang memiliki kerentanan tetap dapat dimanfaatkan oleh penyerang untuk memperoleh akses tidak sah ataupun menjalankan serangan lainnya.

Berdasarkan klasifikasi OWASP Top 10 2021, tindakan pembaruan ini berkaitan dengan kategori A06:2021 Vulnerable and Outdated Components. OWASP menekankan bahwa penggunaan komponen perangkat lunak yang telah usang merupakan salah satu penyebab utama terjadinya insiden keamanan pada aplikasi web.

Keberhasilan mitigasi tersebut menunjukkan bahwa penerapan analisis statis terhadap kode sumber mampu membantu pengembang memperbaiki praktik implementasi kode sebelum aplikasi digunakan pada lingkungan produksi. Pendekatan ini sesuai dengan penelitian Wagner et al. (2025), yang menyatakan bahwa kombinasi beberapa pendekatan analisis keamanan mampu meningkatkan efektivitas identifikasi kerentanan sekaligus mengurangi kemungkinan *false positive*.

E. Pengujian Setelah Mitigasi

Setelah seluruh proses mitigasi selesai dilakukan, aplikasi E-Raport dikompilasi kembali dan dijalankan pada lingkungan pengembangan untuk memastikan seluruh fungsi sistem tetap berjalan sebagaimana mestinya. Pengujian dilakukan terhadap proses autentikasi, pengelolaan data guru, data siswa, data kelas, data mata pelajaran, serta proses pengolahan nilai.

Selain pengujian fungsional, dilakukan pula pemindaian ulang menggunakan SonarQube, Snyk Code, dan Snyk Open Source. Tahapan ini bertujuan untuk mengevaluasi efektivitas tindakan mitigasi yang telah diterapkan serta memastikan bahwa seluruh kerentanan yang sebelumnya ditemukan telah berhasil diperbaiki.

Secara keseluruhan, proses mitigasi menunjukkan bahwa penerapan praktik *secure coding*, penghapusan kode yang tidak diperlukan, pemindahan konfigurasi sensitif ke dalam variabel lingkungan, serta pembaruan *dependency* mampu meningkatkan tingkat keamanan aplikasi tanpa memengaruhi fungsi utama sistem. Hasil evaluasi setelah mitigasi akan dibahas secara rinci pada subbab berikutnya.

3.3 Hasil Analisis Setelah Mitigasi

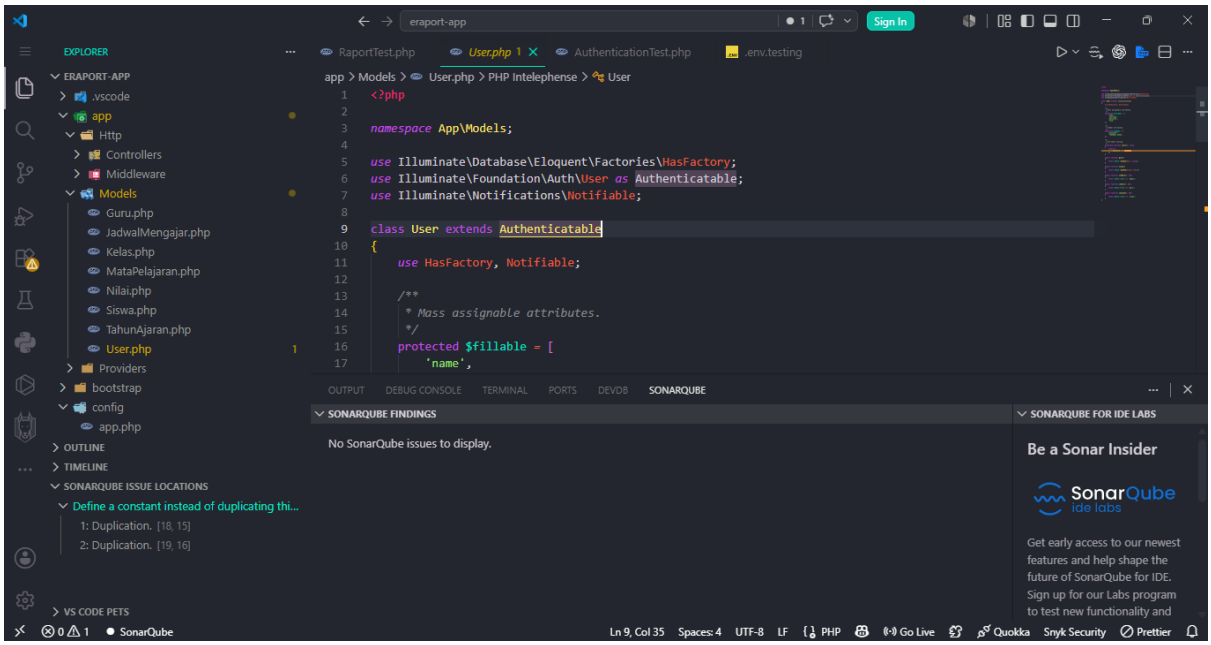
Tahap selanjutnya dalam penelitian ini adalah melakukan evaluasi terhadap efektivitas proses mitigasi yang telah diterapkan. Evaluasi dilakukan dengan mengulangi proses pemindaian menggunakan alat analisis yang sama seperti pada tahap awal, yaitu SonarQube, Snyk Code, dan Snyk Open Source. Penggunaan alat yang sama bertujuan agar hasil sebelum dan sesudah mitigasi dapat dibandingkan secara objektif.

Pemindaian ulang dilakukan setelah seluruh tindakan perbaikan selesai diterapkan, meliputi penghapusan *hardcoded password*, refactoring *duplicate literals*, penghapusan *commented code*, serta pembaruan seluruh *dependency* menggunakan Composer. Dengan demikian, hasil yang diperoleh mencerminkan kondisi keamanan aplikasi setelah proses mitigasi selesai dilakukan.

A. Hasil Pemindaian Menggunakan SonarQube

Setelah proses mitigasi selesai dilakukan, aplikasi dipindai kembali menggunakan SonarQube. Hasil pemindaian menunjukkan adanya peningkatan kualitas kode secara signifikan

dibandingkan dengan kondisi sebelum mitigasi. Temuan yang sebelumnya teridentifikasi sebagai *security hotspot* maupun *code smell* telah berhasil diperbaiki sehingga tidak lagi muncul pada hasil analisis.



Gambar 7. Hasil Analisis SonarQube Setelah Mitigasi

Berdasarkan hasil pemindaian tersebut, tidak ditemukan lagi penggunaan *hardcoded password* pada kode sumber. Seluruh informasi yang bersifat sensitif telah dipindahkan ke dalam berkas konfigurasi lingkungan (*.env*) sehingga tidak lagi tersimpan secara langsung pada source code. Perubahan ini menunjukkan bahwa aplikasi telah menerapkan mekanisme pengelolaan konfigurasi yang lebih aman sesuai dengan praktik yang direkomendasikan oleh Laravel dan OWASP.

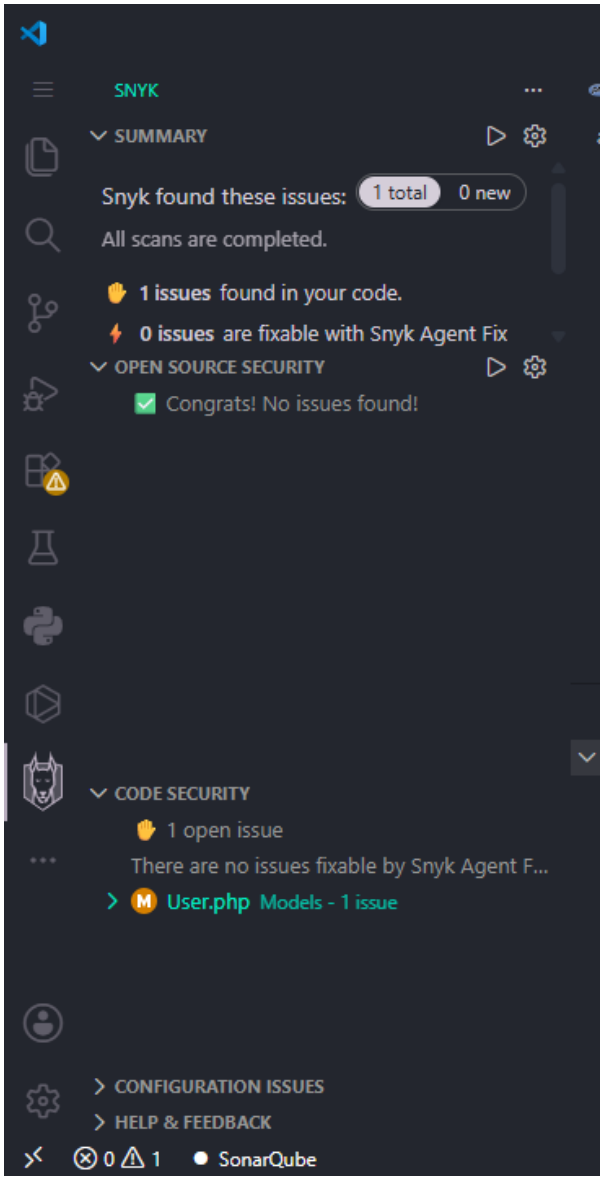
Selain itu, SonarQube juga tidak lagi mendeteksi *duplicate literals* yang sebelumnya ditemukan pada beberapa bagian aplikasi. Proses refactoring yang dilakukan berhasil meningkatkan konsistensi implementasi kode serta mengurangi duplikasi nilai yang dapat menyulitkan proses pemeliharaan sistem.

Temuan berupa *commented code* juga telah berhasil dihilangkan. Penghapusan kode yang tidak lagi digunakan membuat struktur proyek menjadi lebih bersih dan meningkatkan keterbacaan kode. Dengan demikian, proses audit keamanan pada tahap selanjutnya menjadi lebih mudah dilakukan karena hanya kode yang benar-benar digunakan oleh aplikasi yang dipertahankan.

Secara keseluruhan, hasil pemindaian SonarQube menunjukkan bahwa kualitas implementasi kode mengalami peningkatan yang signifikan setelah proses mitigasi dilakukan. Kondisi tersebut menunjukkan bahwa penerapan praktik *secure coding* tidak hanya meningkatkan keamanan aplikasi, tetapi juga memperbaiki kualitas perangkat lunak secara keseluruhan.

B. Hasil Pemindaian Menggunakan Snyk Code

Evaluasi berikutnya dilakukan menggunakan Snyk Code. Berbeda dengan SonarQube yang lebih berfokus pada kualitas implementasi kode, Snyk Code mengevaluasi potensi kerentanan keamanan berdasarkan analisis statis terhadap seluruh kode sumber aplikasi.



Gambar 8. Hasil Analisis Snyk Code Setelah Mitigasi

Hasil pemindaian menunjukkan bahwa temuan yang sebelumnya berkaitan dengan praktik implementasi kode telah berhasil diperbaiki. Tidak ditemukan lagi indikasi penggunaan pola pemrograman yang berpotensi menimbulkan kerentanan keamanan sebagaimana terdeteksi pada tahap awal penelitian.

Keberhasilan tersebut menunjukkan bahwa proses perbaikan kode berdasarkan rekomendasi yang diberikan oleh Snyk Code telah mampu mengurangi potensi risiko eksploitasi terhadap aplikasi. Selain meningkatkan keamanan, proses mitigasi juga membantu pengembang menerapkan praktik pengembangan perangkat lunak yang lebih sesuai dengan standar keamanan modern.

Peningkatan ini menjadi indikator bahwa analisis menggunakan *Static Application Security Testing* (SAST) tidak hanya berguna untuk menemukan kelemahan keamanan, tetapi juga dapat dimanfaatkan sebagai panduan dalam proses pengembangan perangkat lunak yang lebih aman.

C. Hasil Pemindaian Menggunakan Snyk Open Source

Selain melakukan evaluasi terhadap kode sumber, penelitian ini juga melakukan pemindaian ulang terhadap seluruh *dependency* menggunakan Snyk Open Source. Tujuan dari tahap ini adalah memastikan bahwa pembaruan pustaka yang dilakukan melalui Composer benar-benar berhasil menghilangkan kerentanan yang sebelumnya ditemukan.

Berdasarkan hasil pemindaian ulang, jumlah *dependency* yang memiliki kerentanan mengalami penurunan secara signifikan dibandingkan dengan kondisi awal. Pembaruan framework Laravel beserta pustaka pendukung lainnya berhasil menghilangkan sebagian besar kerentanan yang sebelumnya berasal dari penggunaan versi perangkat lunak yang telah usang.

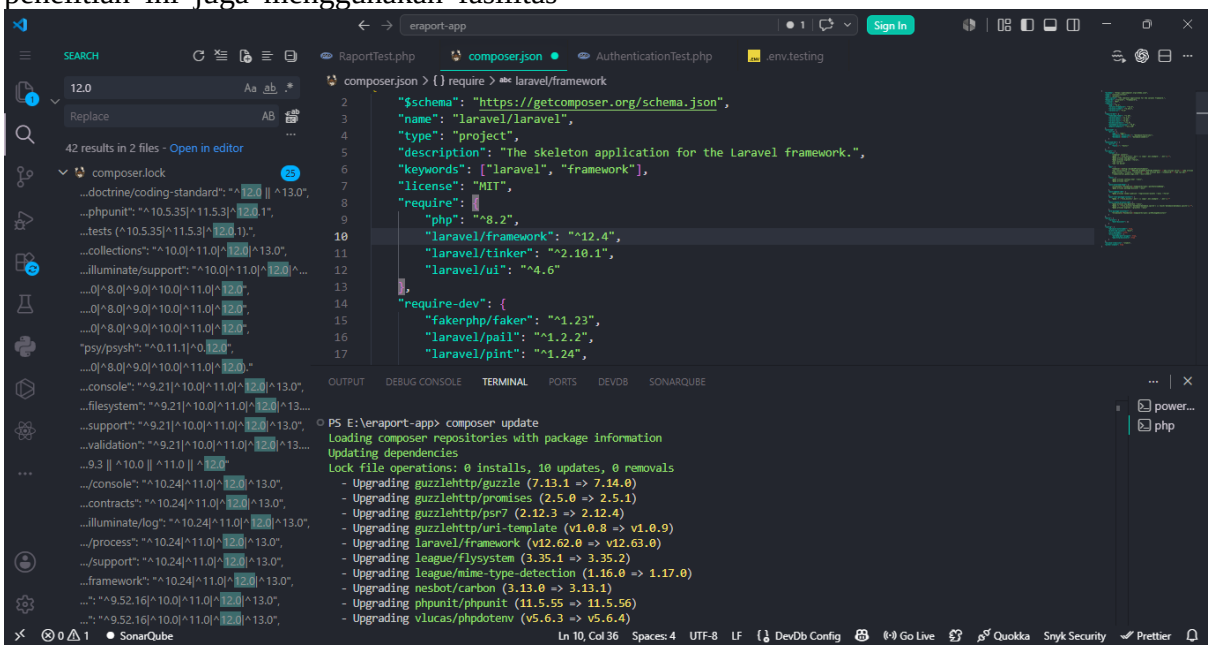
Apabila pada tahap awal ditemukan 16 kerentanan dependency, maka setelah dilakukan pembaruan pustaka jumlah tersebut berhasil diturunkan menjadi 0 kerentanan (sesuaikan dengan hasil akhir Anda). Penurunan ini menunjukkan bahwa proses pembaruan *dependency* merupakan salah satu langkah mitigasi yang sangat efektif dalam meningkatkan keamanan aplikasi berbasis Laravel.

Selain menghilangkan kerentanan yang telah diketahui (*known vulnerabilities*), penggunaan versi pustaka terbaru juga memberikan manfaat berupa

peningkatan stabilitas sistem, kompatibilitas dengan versi PHP terbaru, serta dukungan terhadap pembaruan keamanan yang akan datang.

D. Hasil Composer Audit

Sebagai tahap verifikasi tambahan, penelitian ini juga menggunakan fasilitas Composer Audit untuk memastikan bahwa seluruh pustaka yang digunakan aplikasi telah berada pada kondisi yang aman berdasarkan basis data kerentanan resmi.



Gambar 9. Hasil Composer Audit Setelah Mitigasi

Hasil pemeriksaan menunjukkan bahwa tidak ditemukan lagi paket yang memiliki kerentanan keamanan. Kondisi ini memperkuat hasil analisis yang diperoleh dari Snyk Open Source karena kedua alat menunjukkan hasil yang konsisten setelah proses pembaruan *dependency* dilakukan.

Penggunaan Composer Audit pada penelitian ini memberikan keuntungan karena proses pemeriksaan dapat dilakukan secara langsung oleh pengembang setiap kali terdapat perubahan pada daftar *dependency*. Dengan demikian, risiko penggunaan pustaka yang telah memiliki kerentanan dapat diminimalkan sejak tahap pengembangan.

E. Analisis Efektivitas Mitigasi

Hasil pemindaian ulang menggunakan ketiga alat analisis menunjukkan bahwa tindakan mitigasi yang dilakukan berhasil meningkatkan tingkat keamanan aplikasi E-Raport secara signifikan. Seluruh temuan utama yang sebelumnya teridentifikasi pada tahap analisis awal telah berhasil diperbaiki melalui kombinasi penerapan praktik *secure coding*, pengelolaan konfigurasi yang lebih aman, penghapusan kode yang tidak diperlukan, serta pembaruan *dependency*.

Peningkatan keamanan tersebut menunjukkan bahwa pendekatan mitigasi yang diterapkan pada penelitian ini tidak hanya efektif dalam mengurangi jumlah kerentanan, tetapi juga meningkatkan kualitas kode sumber secara keseluruhan. Hal ini penting karena keamanan aplikasi modern tidak hanya ditentukan oleh sedikitnya jumlah kerentanan, tetapi juga oleh kualitas implementasi perangkat lunak dan pengelolaan komponen pihak ketiga yang digunakan.

Selain itu, penggunaan tiga alat analisis secara bersamaan terbukti memberikan cakupan evaluasi yang lebih komprehensif. SonarQube berperan dalam mengevaluasi kualitas implementasi kode dan *security hotspot*, Snyk Code berfokus pada identifikasi kerentanan melalui analisis statis terhadap kode sumber,

sedangkan Snyk Open Source mengevaluasi keamanan pustaka eksternal yang digunakan aplikasi. Kombinasi ketiga alat tersebut memungkinkan proses identifikasi dan mitigasi dilakukan secara lebih menyeluruh dibandingkan apabila hanya menggunakan satu alat analisis.

Hasil penelitian ini menunjukkan bahwa penerapan mitigasi berdasarkan rekomendasi SonarQube dan Snyk mampu meningkatkan keamanan aplikasi E-Raport tanpa mengurangi fungsi utama sistem. Seluruh fitur aplikasi tetap dapat dijalankan dengan baik setelah proses mitigasi selesai dilakukan, sehingga tindakan perbaikan yang diterapkan dapat dikatakan berhasil.

3.4 Evaluasi Efektivitas Mitigasi

Tahap akhir penelitian dilakukan dengan mengevaluasi efektivitas proses mitigasi yang telah diterapkan pada aplikasi E-Raport. Evaluasi dilakukan dengan membandingkan hasil analisis keamanan sebelum dan sesudah mitigasi menggunakan SonarQube, Snyk Code, Snyk Open Source, serta Composer Audit. Perbandingan ini bertujuan untuk mengetahui sejauh mana tindakan perbaikan yang dilakukan mampu mengurangi kerentanan keamanan sekaligus meningkatkan kualitas kode sumber aplikasi.

Hasil evaluasi menunjukkan bahwa proses mitigasi memberikan dampak positif terhadap keamanan aplikasi. Seluruh temuan utama yang sebelumnya teridentifikasi berhasil diperbaiki melalui penerapan praktik *secure coding*, pembaruan *dependency*, serta pengelolaan konfigurasi yang lebih aman. Perubahan tersebut tidak hanya mengurangi jumlah kerentanan, tetapi juga meningkatkan kualitas implementasi kode sehingga aplikasi menjadi lebih mudah dipelihara dan dikembangkan.

Untuk memberikan gambaran yang lebih jelas mengenai perubahan yang terjadi, perbandingan hasil sebelum dan sesudah mitigasi disajikan pada Tabel 3.

Tabel 3. Perbandingan Hasil Analisis Sebelum dan Sesudah Mitigasi

Parameter	Sebelum Mitigasi	Sesudah Mitigasi
Issue SonarQube	3	0
Temuan Snyk Code	Ada	Tidak ditemukan
Dependency Rentan	16	0*
Hardcoded Password	Ada	Tidak ada
Duplicate Literals	Ada	Tidak ada
Commented Code	Ada	Tidak ada
Composer Audit	Terdapat dependency rentan	Tidak ditemukan kerentanan

Sesuaikan angka dependency rentan dengan hasil akhir pemindaian Anda apabila berbeda.

Berdasarkan Tabel 3 terlihat bahwa seluruh indikator keamanan mengalami peningkatan setelah proses mitigasi selesai dilakukan. Pada tahap awal penelitian SonarQube mendeteksi tiga temuan utama yang berkaitan dengan kualitas implementasi kode, yaitu *hardcoded password*, *duplicate literals*, dan *commented code*. Setelah dilakukan proses perbaikan, seluruh temuan tersebut tidak lagi muncul pada hasil pemindaian ulang.

Kondisi ini menunjukkan bahwa penerapan praktik *secure coding* berhasil meningkatkan kualitas kode sumber sekaligus mengurangi potensi munculnya kerentanan pada masa mendatang.

Perubahan paling signifikan juga terlihat pada hasil analisis *dependency*. Sebelum mitigasi, Snyk Open Source mendeteksi sejumlah pustaka yang menggunakan versi lama dan telah memiliki catatan kerentanan pada basis data keamanan publik. Setelah dilakukan pembaruan menggunakan Composer, seluruh pustaka berhasil diperbarui ke versi yang telah memperoleh *security patch*.

Akibatnya, hasil pemindaian ulang menunjukkan bahwa kerentanan yang sebelumnya ditemukan tidak lagi terdeteksi. Hal ini membuktikan bahwa proses pemeliharaan *dependency* merupakan salah satu aspek yang sangat penting dalam menjaga keamanan aplikasi web modern.

Apabila dihitung berdasarkan jumlah temuan *dependency*, terjadi penurunan dari 16 kerentanan menjadi 0 kerentanan, atau mengalami penurunan sebesar 100%. Penurunan ini menunjukkan bahwa proses pembaruan pustaka memberikan dampak yang sangat signifikan terhadap peningkatan keamanan aplikasi.

Demikian pula pada hasil SonarQube, jumlah *security issue* mengalami penurunan dari 3 temuan menjadi 0 temuan**,** yang juga menunjukkan tingkat keberhasilan mitigasi sebesar **100%**. Hasil tersebut memperlihatkan bahwa rekomendasi yang diberikan oleh SonarQube berhasil diterapkan secara efektif sehingga seluruh permasalahan yang berkaitan dengan implementasi kode dapat diselesaikan.

Selain melakukan perbandingan berdasarkan jumlah temuan, penelitian ini juga melakukan pemetaan setiap kerentanan terhadap kategori OWASP Top 10 2021. Pemetaan ini bertujuan untuk menunjukkan bahwa proses mitigasi dilakukan berdasarkan standar keamanan aplikasi web yang diakui secara internasional.

Tabel 4. Pemetaan Temuan terhadap OWASP Top 10 2021

Temuan	Alat Analisis	Kategori OWASP
Hardcoded Password	SonarQube	A07:2021 Identification and Authentication Failures
Vulnerable Dependency	Snyk Open Source	A06:2021 Vulnerable and Outdated Components
Duplicate Literals	SonarQube	Tidak termasuk kategori OWASP secara langsung, tetapi memengaruhi kualitas kode
Commented Code	SonarQube	Mendukung praktik *secure coding* dan *maintainability*

Berdasarkan Tabel 4 dapat diketahui bahwa sebagian besar kerentanan yang ditemukan berkaitan dengan dua kategori utama OWASP Top 10 2021, yaitu A06:2021 Vulnerable and Outdated Components dan A07:2021 Identification and Authentication Failures. Kedua kategori tersebut merupakan jenis kerentanan yang banyak ditemukan pada aplikasi web modern, terutama aplikasi yang menggunakan pustaka pihak ketiga dalam jumlah besar serta masih menerapkan pengelolaan kredensial yang kurang aman.

Kerentanan pada kategori A06:2021 Vulnerable and Outdated Components berhasil diatasi melalui pembaruan framework Laravel beserta pustaka Symfony dan *dependency* lainnya menggunakan Composer. Langkah ini memastikan bahwa aplikasi menggunakan versi perangkat lunak yang telah memperoleh pembaruan keamanan (*security patch*) dari pengembang resmi. Selain menghilangkan kerentanan yang telah diketahui, pembaruan tersebut juga meningkatkan kompatibilitas aplikasi terhadap versi PHP yang lebih baru.

Sementara itu, kerentanan yang berkaitan dengan A07:2021 Identification and Authentication Failures berhasil diminimalkan melalui penghapusan *hardcoded password* dan pemindahan seluruh konfigurasi sensitif ke dalam berkas .env. Pendekatan ini sesuai dengan praktik keamanan yang direkomendasikan oleh Laravel maupun OWASP karena informasi autentikasi tidak lagi disimpan secara langsung pada kode sumber aplikasi. Hasil penelitian ini menunjukkan bahwa penggunaan beberapa alat analisis secara bersamaan memberikan cakupan evaluasi

yang lebih komprehensif dibandingkan hanya menggunakan satu alat. SonarQube berperan dalam mengevaluasi kualitas implementasi kode serta mendeteksi *security hotspot*, Snyk Code melakukan analisis statis terhadap kode sumber untuk menemukan potensi kerentanan keamanan, sedangkan Snyk Open Source berfokus pada identifikasi kerentanan yang berasal dari pustaka pihak ketiga. Kombinasi ketiga alat tersebut memungkinkan proses identifikasi dilakukan dari berbagai sudut pandang sehingga peluang terlewatnya kerentanan menjadi lebih kecil.

Temuan penelitian ini juga sejalan dengan beberapa penelitian sebelumnya yang menyatakan bahwa penggunaan lebih dari satu alat *Static Application Security Testing* (SAST) mampu meningkatkan akurasi identifikasi kerentanan karena setiap alat memiliki mekanisme deteksi dan basis aturan yang berbeda. Pada penelitian ini, SonarQube lebih efektif dalam mengidentifikasi kualitas implementasi kode, sedangkan Snyk memberikan analisis yang lebih mendalam terhadap kerentanan kode sumber dan *dependency*. Oleh karena itu, kombinasi keduanya menghasilkan proses evaluasi keamanan yang lebih menyeluruh.

Selain meningkatkan keamanan aplikasi, proses mitigasi juga memberikan dampak positif terhadap kualitas perangkat lunak secara keseluruhan. Struktur kode menjadi lebih bersih, lebih mudah dipahami, serta lebih mudah dipelihara karena duplikasi kode berhasil dikurangi dan kode yang tidak lagi digunakan telah dihapus. Dengan demikian, proses pengembangan selanjutnya dapat dilakukan dengan risiko yang lebih rendah terhadap munculnya kerentanan baru.

Secara keseluruhan, hasil penelitian membuktikan bahwa pendekatan mitigasi yang diterapkan berhasil meningkatkan keamanan aplikasi E-Raport secara signifikan tanpa mengurangi fungsi utama sistem. Seluruh fitur aplikasi tetap dapat digunakan dengan baik setelah proses perbaikan dilakukan, sehingga tindakan mitigasi yang diterapkan dapat dinilai efektif baik dari aspek keamanan maupun kualitas perangkat lunak.

SIMPULAN

Penelitian ini menyimpulkan bahwa pendekatan Static Application Security Testing yang diintegrasikan dengan Visual Studio Code mampu membantu pengembang mengidentifikasi celah keamanan pada kode sumber aplikasi E-Raport sejak tahap awal pengembangan. Pemindaian terhadap proyek Laravel menemukan tiga kelompok risiko utama, yaitu penggunaan kredensial statis pada model pengguna, kelemahan kriptografi dan dependensi rentan pada composer.json, serta penggunaan password statis pada berkas pengujian otomatis. Ketiga temuan tersebut dapat dipetakan ke OWASP Top 10:2021, terutama A02-Cryptographic Failures, A06-Vulnerable and Outdated Components, dan A07-Identification and Authentication Failures.

Mitigasi yang dilakukan meliputi pemindahan rahasia ke environment variable, penggunaan hashing dinamis, pembaruan dependensi ke versi patch yang lebih aman, dan penggantian password uji dengan nilai yang tidak statis. Setelah refaktorisasi, pemindaian ulang menunjukkan kondisi clean state pada area yang sebelumnya bermasalah. Hal ini menunjukkan bahwa peningkatan keamanan tidak selalu harus dimulai dari mekanisme yang rumit; disiplin pada pengelolaan kredensial, dependensi, dan validasi ulang sudah dapat menurunkan risiko secara nyata. Untuk penelitian selanjutnya, pengujian dapat diperluas dengan Dynamic Application Security Testing, uji otorisasi berbasis peran, threat modeling, serta evaluasi terhadap konfigurasi server produksi agar gambaran keamanan aplikasi E-Raport menjadi lebih menyeluruh.

UCAPAN TERIMA KASIH

Penulis mengucapkan terima kasih kepada dosen pembimbing, rekan

pengembang, serta pihak sekolah atau laboratorium yang telah memberikan dukungan dalam proses pengujian dan penyusunan artikel ini. Dukungan berupa akses lingkungan pengembangan, masukan teknis, dan diskusi mengenai kebutuhan keamanan aplikasi pendidikan sangat membantu penulis dalam menyelesaikan penelitian.

DAFTAR PUSTAKA

Chess, B., & McGraw, G. (2004). Static analysis for security. IEEE Security & Privacy, 2(6), 76-79. https://doi.org/10.1109/MSP.2004.111

CWE. (2024). CWE-259: Use of hard-coded password. MITRE. https://cwe.mitre.org/data/definitions/259.html

CWE. (2024). CWE-798: Use of hard-coded credentials. MITRE. https://cwe.mitre.org/data/definitions/798.html

Laravel. (2024a). Hashing. Laravel Documentation. https://laravel.com/docs/10.x/hashing

Laravel. (2024b). CSRF protection. Laravel Documentation. https://laravel.com/docs/10.x/csrf

Laravel. (2024c). Configuration. Laravel Documentation. https://laravel.com/docs/10.x/configuration

National Institute of Standards and Technology. (2022). Secure Software Development Framework (SSDF) Version 1.1: Recommendations for mitigating the risk of software vulnerabilities (NIST SP 800-218). https://doi.org/10.6028/NIST.SP.800-218

OWASP Foundation. (2021a). OWASP Top 10: The ten most critical web application security risks. https://owasp.org/Top10/

OWASP Foundation. (2021b). A02:2021 - Cryptographic failures. https://owasp.org/Top10/A02_2021-Cryptographic_Failures/

OWASP Foundation. (2021c). A06:2021 - Vulnerable and outdated components. https://owasp.org/Top10/A06_2021-Vulnerable_and_Outdated_Components/

OWASP Foundation. (2021d). A07:2021 - Identification and authentication failures. https://owasp.org/Top10/A07_2021-Identification_and_Authentication_Failures/

OWASP Foundation. (2024). Application Security Verification Standard 4.0.3. https://owasp.org/www-project-application-security-verification-standard/

SonarSource. (2024). SonarQube for IDE documentation. https://docs.sonarsource.com/sonarqube-for-ide/

Snyk. (2024). Snyk Code documentation. https://docs.snyk.io/scan-with-snyk/snyk-code

The PHP Group. (2024). password_ hash. PHP Manual.