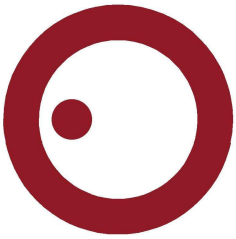


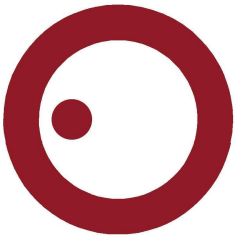
escola superior de tecnologia e gestão
instituto politécnico de leiria

Instituto Politécnico de Leiria
Escola Superior de Tecnologia e Gestão
Departamento de Engenharia Informática
Mestrado em Eng.ª Informática – Cibersegurança e Informática Forense

VULNFUSION: PLATAFORMA DE INTEGRAÇÃO DE FERRAMENTAS *OPEN-SOURCE* SAST

EDGAR EMANUEL RAIMUNDO ANDRADE

Leiria, Setembro de 2025



escola superior de tecnologia e gestão
instituto politécnico de leiria

Instituto Politécnico de Leiria
Escola Superior de Tecnologia e Gestão
Departamento de Engenharia Informática
Mestrado em Eng.ª Informática – Cibersegurança e Informática Forense

VULNFUSION: PLATAFORMA DE INTEGRAÇÃO DE FERRAMENTAS *OPEN-SOURCE* SAST

EDGAR EMANUEL RAIMUNDO ANDRADE
Número: 2230054

Projeto realizado sob orientação do Professor Doutor Paulo Jorge Gonçalves Loureiro (paulo.loureiro@ipleiria.pt) e Professor Doutor Sílvio Priem Mendes (smendes@ipleiria.pt).

Leiria, Setembro de 2025

AGRADECIMENTOS

Em primeiro lugar, gostaria de agradecer ao Professor Doutor Paulo Loureiro e ao Professor Doutor Sílvio Priem Mendes pelo tempo dedicado, ajuda prestada e também pela constante troca de ideias que contribuíram significativamente para a melhoria deste projeto.

Pela partilha de conhecimento, apoio e camaradagem, gostaria de agradecer aos meus colegas do mestrado de Cibersegurança e Informática Forense, com quem tive o privilégio de aprender e crescer.

Quero ainda deixar uma palavra de agradecimento a todos os professores que lecionaram as unidades curriculares do mestrado, pela fantástica capacidade de transmitir conhecimento e pela criação de um ambiente excecional, dentro e fora da sala de aula.

RESUMO

Ciclos de desenvolvimento acelerados e o uso de bibliotecas de terceiros aumentam a probabilidade de existirem lacunas de segurança no código, agravada pela insuficiente formação em segurança dos programadores. A análise de segurança manual é lenta e cara, o que torna as ferramentas de análise de segurança estática, *static application security testing* (SAST), essenciais para identificar vulnerabilidades de forma eficiente.

Contudo, as ferramentas SAST geram muitos falsos positivos, exigindo verificação manual. Este projeto propõe uma plataforma, intitulada VulnFusion, que integra múltiplas ferramentas SAST para melhorar a robustez da análise de segurança, e que utiliza técnicas de inteligência artificial (IA) para enriquecer os resultados obtidos.

A plataforma VulnFusion visa reduzir os falsos positivos e aumentar a eficiência na deteção de vulnerabilidades e auxiliar na prioritização dos esforços de mitigação, adaptando-se às necessidades dos programadores e organizações.

O presente documento inclui uma breve revisão de ferramentas SAST *open-source* existentes, o desenvolvimento da plataforma VulnFusion e dos processos de agregação de resultados e enriquecimento dos mesmos, e apresenta os testes realizados e os resultados obtidos.

Os resultados demonstraram uma melhoria na cobertura e precisão da deteção de vulnerabilidades face à utilização de uma única ferramenta, com ganhos no F1-score para categorias como *command injection* (47,8%) e *SQL injection* (36,5%) após a integração de múltiplas ferramentas SAST e enriquecimento por IA.

ABSTRACT

Accelerated development cycles and the use of third-party libraries increase the likelihood of security gaps in the code, exacerbated by the insufficient security training of developers. Manual security analysis are slow and expensive, making static application security testing (SAST) tools essential for efficiently identifying vulnerabilities.

However, SAST tools generate many false positives, requiring manual verification. This project proposes a platform, titled VulnFusion, that integrates multiple SAST tools to improve the robustness of security analysis and uses artificial intelligence (AI) techniques to enrich the results produced.

The platform VulnFusion aims to reduce false positives, increase the efficiency of vulnerability detection, and assist in prioritizing mitigation efforts, adapting to the needs of developers and organizations.

This document includes a brief review of existing open-source SAST tools, the development of the platform and the processes for aggregating and enriching results, and presents the tests performed and the results obtained.

The results obtained demonstrated an improvement in both coverage and accuracy of vulnerability detection compared to the use of a single tool, with F1-score gains for categories such as command injection (47.8%) and SQL injection (36.5%) following the integration of multiple SAST tools and AI-based enrichment.

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

API	Application Programming Interface.
CE	Community Edition.
CERT	Computer Emergency Response Team.
CSV	Comma-Separated Values.
CWE	Common Weakness Enumeration.
ETL	Extract, Transform and Load.
FN	Falso Negativo (*False Negative*).
FP	Falso Positivo (*False Positive*).
GUI	Graphical User Interface.
HTML	HyperText Markup Language.
IA	Inteligência Artificial.
JDK	Java Development Kit.
JSON	JavaScript Object Notation.
NVD	National Vulnerability Database.
SARIF	Static Analysis Results Interchange Format.
SAST	Static Application Security Testing.
SEI	Software Engineering Institute.
SQI	Security and Quality Impact.
TP	True Positive.

VP Verdadeiro Positivo.

XML Extensible Markup Language.

XSS Cross-Site Scripting.

1

INTRODUÇÃO

Num contexto de desenvolvimento de software em constante evolução, a segurança continua a ser uma preocupação crítica, especialmente à medida que as aplicações se tornam mais complexas e distribuídas. As vulnerabilidades presentes no código podem comprometer a integridade, confidencialidade e disponibilidade dos sistemas, sendo por isso essencial a adoção de práticas proativas de análise e mitigação.

As ferramentas de Static Application Security Testing (SAST) oferecem uma abordagem eficaz para a deteção precoce de vulnerabilidades, ao analisarem o código fonte sem necessidade de execução. Embora as soluções comerciais apresentem capacidades avançadas de deteção e funcionalidades como análise assistida por inteligência artificial, o seu custo elevado torna-as inacessíveis para programadores independentes e pequenas empresas.

Com o objetivo de colmatar esta limitação, este trabalho propõe a plataforma VulnFusion, modular e extensível, que integra ferramentas SAST de código aberto e serviços de inteligência artificial, permitindo realizar análises de segurança em projetos de diferentes linguagens sem incorrer nos custos das soluções comerciais. A VulnFusion unifica os resultados de múltiplas ferramentas, normaliza os achados e aplica técnicas de enriquecimento por IA, facilitando a identificação de vulnerabilidades críticas e a priorização dos esforços de mitigação.

A eficácia da abordagem proposta foi avaliada através de testes com conjuntos de referência, como o Juliet Test Suite, e projetos reais como o VulnerableApp, demonstrando que a agregação multi-ferramenta e o enriquecimento assistido por IA contribuem para melhorar a deteção de vulnerabilidades e orientar as decisões de correção.

1.1 OBJETIVOS DO TRABALHO DE INVESTIGAÇÃO

O trabalho de investigação apresentado neste documento teve como base quatro objetivos principais, que definiram o processo de desenvolvimento. Estes objetivos são:

- **Desenvolver uma plataforma de integração de ferramentas SAST *open-source***: criar uma plataforma modular que permita realizar a análise de segurança de software através da execução coordenada de diversas ferramentas SAST *open-source*, sem custos de licenciamento.
- **Aumentar a precisão na deteção de vulnerabilidades**: combinar os pontos fortes das diferentes ferramentas utilizadas, de forma a melhorar a cobertura da análise e reduzir a ocorrência de falsos positivos.
- **Implementar uma metodologia de agregação de resultados**: desenvolver um processo de união dos relatórios produzidos pelas ferramentas SAST num relatório único, que forneça ao utilizador informação de forma estruturada e de fácil interpretação.
- **Integrar inteligência artificial**: utilização de técnicas de inteligência artificial para reduzir a ocorrência de falsos positivos e auxiliar na prioritização dos esforços de mitigação.

1.2 METODOLOGIA DE DESENVOLVIMENTO

O desenvolvimento da plataforma VulnFusion seguiu uma metodologia estruturada, organizada em cinco fases distintas:

- **Revisão bibliográfica e levantamento de ferramentas SAST**: estudo de artigos relacionados e análise de diversas ferramentas SAST *open-source*, considerando critérios como popularidade, suporte a linguagens, funcionalidades e facilidade de integração.
- **Seleção de ferramentas SAST**: escolha de quatro ferramentas SAST e desenvolvimento de contentores *docker* para a execução das ferramentas selecionadas.
- **Desenvolvimento da plataforma**: definição da arquitetura de plataforma, criação dos processos responsáveis pela execução das ferramentas SAST selecionadas, normalização e união dos relatórios individuais e criação do interface gráfica.
- **Integração de inteligência artificial**: utilização da *Application Programming Interface* (API) da OpenAI para o enriquecimento dos relatórios produzidos, com a atribuição de classificações *Common Weakness Enumeration* (CWE), cálculo de métricas de *likelihood* e *risk-score*, e fornecimento de recomendações de mitigação.

- **Validação experimental**: realização de testes com o objetivo de medir a eficácia da plataforma, desde a atribuição da classificação CWE à melhoria da cobertura e precisão da deteção de vulnerabilidades e na redução de falsos positivos.

1.3 ESTRUTURA DO RELATÓRIO

O presente documento encontra-se organizado em seis capítulos. Com a exceção do capítulo atual, os restantes capítulos encontram-se descritos de seguida:

- Capítulo 2 - Trabalho Relacionado: aborda artigos e informação relevantes para o desenvolvimento do trabalho descrito no presente documento;
- Capítulo 3 - Desenvolvimento: apresenta as ferramentas SAST selecionadas e a arquitetura da plataforma VulnFusion;
- Capítulo 4 - Resultados: descreve as metodologias de teste implementadas e os resultados obtidos;
- Capítulo 5 - Conclusões e trabalho futuro, onde se apresenta diversas melhorias identificadas e novas funcionalidades a adicionar.

2

TRABALHO RELACIONADO

O presente capítulo tem como objetivo introduzir o trabalho desenvolvido por outros autores, utilizado como ponto de partida para o desenvolvimento do trabalho descrito neste documento.

2.1 ESTUDOS COMPARATIVOS DE FERRAMENTAS SAST

Diversos autores realizaram análises comparativas para avaliar a eficácia das ferramentas SAST.

O trabalho apresentado em [1], por Matteo Esposito, tem como objetivo apresentar um *benchmark* que pode ser utilizado para avaliar o mecanismo de deteção das ferramentas SAST. Os autores utilizaram a versão JAVA do Juliet Test Suite v1.3 (JTS), que contém casos de teste organizados em 112 diferentes CWEs, para avaliar seis ferramentas SAST *open-source* e duas comerciais.

Os resultados dos testes obtidos demonstraram que a taxa de deteção real das ferramentas SAST foi significativamente inferior ao esperado (inferior a 50%), e que uma única ferramenta conseguiu identificar apenas 11% dos CWEs. Os autores observaram também que cada ferramenta SAST apresentava melhor desempenho de deteção num CWE específico. Através das métricas obtidas, os autores salientam que as ferramentas SAST apresentaram um baixo número de falsos positivos, sendo que a sua principal falha reside nos falsos negativos.

Os autores deixam várias recomendações, das quais destacamos as seguintes:

- O desempenho de uma ferramenta deve basear-se em resultados empíricos e não na documentação oficial.
- Deve-se usar várias ferramentas SAST para identificar diferentes tipos de vulnerabilidades.
- As ferramentas SAST devem ser escolhidas com base no CWE e no aspeto de precisão relevante para o projeto-alvo.

Em [2], os autores realizaram uma análise comparativa de várias ferramentas SAST com o objetivo de calcular e analisar o seu desempenho, bem como determinar as vantagens e desvantagens associadas a estas ferramentas. Este estudo abrange 16 aplicações web distintas com vulnerabilidades conhecidas, desenvolvidas com tecnologias diversas, e 11 ferramentas SAST *open-source* passíveis de execução local. Algumas das ferramentas utilizadas são específicas de uma tecnologia, enquanto outras conseguem detetar vulnerabilidades em múltiplas tecnologias.

Devido às diferenças na forma como cada ferramenta apresenta os resultados da análise, os autores desenvolveram um processo *Extract, Transform and Load* (ETL) independente, capaz de extrair os resultados individuais e transformar os dados num único formato, o que permitiu uma comparação mais clara de cada ferramenta.

Os autores concluem que o processo de seleção de uma ferramenta é complexo, uma vez que o desempenho pode variar consoante o contexto, e que, em geral, as ferramentas direcionadas a uma tecnologia específica superam aquelas que suportam várias.

2.2 COMBINAÇÃO E COMPLEMENTARIDADE DE FERRAMENTAS

A combinação de diferentes ferramentas SAST tem sido explorada na literatura como uma forma de minimizar as limitações individuais de cada ferramenta SAST e melhorar a cobertura e a precisão de deteção de vulnerabilidades. Esta temática revela-se particularmente relevante para o trabalho desenvolvido no presente documento.

O trabalho apresentado em [3], procurou compreender as limitações de quatro ferramentas SAST quando executadas no conjunto de dados SAP, proveniente da *National Vulnerability Database* (NVD), que consiste em código de produção com vulnerabilidades conhecidas, de modo a representar melhor o desempenho das ferramentas num ambiente real.

Os autores determinaram que a taxa de deteção individual das ferramentas SAST selecionadas estava entre 11,2% e 26,5%. Esse valor aumentou para 38,38% quando as ferramentas foram combinadas. A decomposição dos resultados por CWE revelou que algumas vulnerabilidades não são amplamente detetadas e que certas ferramentas detetam exclusivamente determinados tipos de vulnerabilidades.

A razão pela qual algumas vulnerabilidades não foram detetadas pelas ferramentas, segundo os autores, deveu-se a uma taxonomia incompleta de regras. Ao editar a

configuração do Semgrep, através da adição de novas regras, os autores observaram um aumento na taxa de deteção desta ferramenta de 15,3% para 44,7%.

O artigo conclui que as ferramentas SAST podem complementar-se mutuamente, mas o utilizador deve esforçar-se por compreender o funcionamento destas, selecionar as mais adequadas ao projeto-alvo e investir tempo na sua configuração para obter melhores resultados.

O trabalho desenvolvido em [4] estuda a combinação de várias ferramentas SAST como forma de melhorar a deteção de vulnerabilidades em diferentes cenários realistas de desenvolvimento, tendo em conta a criticidade da aplicação em desenvolvimento e o orçamento de segurança disponível.

Este estudo baseou-se na análise de 134 plugins WordPress com cinco ferramentas SAST gratuitas, focando-se na deteção de vulnerabilidades de SQL injection e cross-site scripting. Os cenários de desenvolvimento considerados foram:

- Qualidade máxima – devido à elevada criticidade da aplicação, cada vulnerabilidade não detetada pode representar um problema significativo.
- Alta qualidade – podem passar despercebidas algumas vulnerabilidades não triviais.
- Qualidade média – podem faltar vulnerabilidades em troca de reduzir os falsos positivos.
- Qualidade mínima – cada falso positivo é motivo de preocupação devido a limitações orçamentais.

Para cada cenário apresentado, os autores definem uma métrica específica para avaliar o desempenho das ferramentas SAST, de forma a adequar o objetivo da deteção aos recursos disponíveis para tratar os resultados.

A combinação dos resultados produzidos pelas diferentes ferramentas consiste, primeiro, em caracterizar os resultados individuais para determinar o número de positivos e negativos; em seguida, combinar os resultados, descartando-se os verdadeiros positivos (VP) e falsos positivos (FP) duplicados; depois, criar a matriz de confusão para o resultado combinado; e, por fim, determinar as métricas e a classificação com base nessa matriz de confusão.

Os autores concluem que combinar os resultados de várias ferramentas SAST gratuitas nem sempre obriga a um melhor desempenho na deteção de vulnerabilidades, mas que, em princípio, a combinação de várias ferramentas traz benefícios devido à natureza complementar que estas podem apresentar.

Em geral, os autores defendem que, à medida que o número de ferramentas aumenta, o acréscimo na deteção de novas vulnerabilidades e de novos falsos positivos também aumenta, mas de forma mais reduzida e a ritmos diferentes.

2.3 REDUÇÃO DE FALSOS POSITIVOS COM TÉCNICAS DE IA

Uma área de investigação relevante aborda a aplicação de técnicas de inteligência artificial para reduzir falsos positivos e apoiar a priorização dos resultados das ferramentas SAST.

Em [5], o objetivo de Lori Flynn, foi facilitar o esforço necessário para triagem de alertas e priorização dos avisos fornecidos pelas ferramentas SAST. Para tal, os autores desenvolveram modelos de classificação capazes de prever se um alerta correspondia a um verdadeiro ou a um falso positivo.

O processo descrito pelos autores consistiu, primeiro, em inspecionar o código-fonte com as ferramentas SAST selecionadas. Em seguida, os relatórios gerados pelas diferentes ferramentas e o próprio código-fonte foram remetidos para uma versão melhorada da ferramenta SCALe, que converte os diversos relatórios de uma única base de código num formato de dados comum e fornece ainda o mapeamento dos alertas para violações das SEI CERT *Coding Rules* e outras métricas de código. Esses artefatos foram depois analisados por auditores humanos, que classificaram os alertas em verdadeiros e falsos positivos para criar o conjunto de dados de treino.

Os autores utilizaram quatro técnicas de classificação, Regressão Logística Lasso, CART, *Random Forest* e *Extreme Gradient Boosting*, para gerar e comparar diferentes modelos. Estes modelos revelaram um elevado nível de precisão, sendo o valor mais baixo de 87%, o que demonstra que a abordagem dos autores pode ter um impacto significativo nos recursos necessários para resolver os alertas.

Os dados para desenvolvimento dos classificadores provêm de arquivos de 19 bases de código auditadas pelo CERT e de informações de três organizações colaboradoras. A falta de dados classificados constituiu uma das limitações encontradas pelos autores ao longo do estudo.

O trabalho [6], apresentado por Ribeiro, teve como objetivo desenvolver um modelo preditivo para reforçar a eficácia das ferramentas SAST, sem exigir um exame direto do código-fonte. A abordagem proposta pelos autores aborda o problema das elevadas taxas de falsos positivos apresentadas por estas ferramentas.

Este estudo foi desenvolvido com recurso ao Juliet Test Suite (v1.2) e a três ferramentas SAST de código aberto. Ao analisar uma versão selecionada (*pruned*) do pacote de testes com as ferramentas escolhidas, os autores recolheram um número substancial de alertas. Estes alertas surgiam em formatos distintos consoante o relatório gerado pela ferramenta, pelo que foram convertidos para um formato unificado, permitindo extrair características relevantes, como o número de alertas no mesmo ficheiro, o nome da ferramenta, os níveis de redundância e a categoria do alerta.

Os autores recorreram ao algoritmo AdaBoost para treinar vários classificadores fracos, que foram depois combinados numa solução mais robusta. Esta abordagem permitiu classificar e hierarquizar os alertas, atribuindo melhores posições aos que tinham maior probabilidade de corresponder a falhas reais. O modelo desenvolvido pelos autores alcançou uma precisão de 0,8, sem necessidade de acesso direto ao código-fonte, tornando-se num modelo genérico passível de aplicação a diferentes projetos de software.

Este estudo demonstra um método prático para aumentar a utilidade das ferramentas de análise estática, potenciando a sua adoção generalizada nos processos de desenvolvimento de software.

2.4 FERRAMENTAS SAST DE REFERÊNCIA

Para além das soluções *open-source*, diversas ferramentas comerciais de análise estática de segurança são amplamente reconhecidas na indústria. Entre estas destacam-se SonarQube, Fortify e Checkmarx, frequentemente utilizadas como referência em estudos comparativos [7] [1]. Estas ferramentas destacam-se pelo suporte empresarial, cobertura de múltiplas linguagens e funcionalidades avançadas, como utilização de inteligência artificial, sendo consideradas *benchmarks* para avaliação de novas soluções.

O SonarQube é uma das plataformas mais difundidas, com suporte a múltiplas linguagens e integração nativa em *pipelines* de desenvolvimento, sendo frequentemente usado como métrica de qualidade de código e segurança. O Fortify e o Checkmarx são soluções comerciais robustas, que oferecem extensas bases de regras e funcionalidades avançadas de integração com processos DevSecOps, incluindo relatórios de conformidade e mecanismos de priorização de vulnerabilidades.

Apesar da sua relevância, o custo associado a estas ferramentas constitui uma limitação significativa para programadores independentes e pequenas organizações. Assim, estas soluções não se alinham com o objetivo central deste trabalho, que é a avaliação e integração de ferramentas *open-source*, de modo a disponibilizar uma alternativa acessível e extensível à comunidade.

2.5 NORMALIZAÇÃO E FUSÃO DE RELATÓRIOS

A literatura destaca a necessidade de normalizar e fundir os relatórios provenientes de diferentes ferramentas SAST, de forma a facilitar a agregação e análise dos resultados. Neste contexto, o *Static Analysis Results Interchange Format* (SARIF), definido pela OASIS, surge como um padrão que permite a uniformização de resultados previamente heterogéneos. A adoção deste formato tem vindo a crescer na comunidade DevSecOps, precisamente pela sua capacidade de suportar a agregação de múltiplas fontes de análise [8].

Kummita e Piskachev [9], exploraram a conversão dos resultados provenientes do CogniCrypt para o formato SARIF. Embora os autores não discutam diretamente o impacto de apresentar os resultados em um formato padrão, o trabalho apresentado ilustra o potencial do SARIF para facilitar a interoperabilidade e a comparação entre ferramentas.

O artigo *Using SARIF to Automate Vulnerability Remediation Tracking* de Jessica Grider [10] publicado na plataforma LinkedIn, descreve a experiência da empresa Policygenius na aplicação do SARIF para automatizar o processo de mitigação de vulnerabilidades. Embora não se trate de uma publicação académica, este caso ilustra de forma concreta a adoção do SARIF em *pipelines* DevSecOps, incluindo a normalização de resultados de múltiplas ferramentas SAST e a criação automática de *tickets* no Jira e alertas via Slack. Este exemplo demonstra que a utilização de um formato padrão como o SARIF pode efetivamente suportar a integração e uniformização de diferentes ferramentas SAST no acompanhamento contínuo e na correção de vulnerabilidades em ambientes empresariais.

A plataforma GitHub também suporta a fusão automática de múltiplos relatórios SARIF através da sua ferramenta de análise estática CodeQL, recorrendo à funcionalidade *merge-results* [11]. Isto demonstra a relevância prática do SARIF em cenários de integração contínua, permitindo consolidar resultados de diferentes execuções ou ferramentas num único relatório, facilitando a gestão centralizada de vulnerabilidades.

Estes exemplos demonstram que a fusão de relatórios provenientes de diferentes ferramentas SAST, suportada por um formato universal como o SARIF, constitui uma estratégia já reconhecida no contexto de DevSecOps.

2.6 SÍNTESE

O presente capítulo realiza uma revisão da literatura e de estudos existentes sobre ferramentas SAST, abordando comparações de desempenho, combinação de ferramentas, técnicas de redução de falsos positivos e normalização de relatórios. Esta revisão demonstrou que a utilização individual de uma ferramenta SAST apresenta limitações, e que a combinação de múltiplas ferramentas, aliada a técnicas de IA, pode melhorar a cobertura e precisão na deteção de vulnerabilidades.

O Capítulo 3 descreve o desenvolvimento da plataforma VulnFusion, incluindo a seleção das ferramentas SAST, a arquitetura da plataforma, os processos de integração e enriquecimento dos resultados, assente nas abordagens e recomendações identificadas na revisão bibliográfica apresentada neste capítulo.

3

DESENVOLVIMENTO

A realização deste projeto teve como objetivo desenvolver a plataforma VulnFusion, que permita realizar uma análise de segurança de projetos informáticos, com recurso a múltiplas ferramentas SAST e onde os resultados são enriquecidos com a aplicação de técnicas de inteligência artificial.

Serve o presente capítulo para descrever as ferramentas SAST selecionadas e o respetivo processo de seleção, e detalhar a arquitetura da VulnFusion, bem como as opções tomadas no processo de desenvolvimento.

3.1 FERRAMENTAS SAST

Um dos objetivos da plataforma VulnFusion apresentada neste documento consiste em permitir aos utilizadores analisar projetos desenvolvidos com recurso a tecnologias distintas, sem ser necessário a utilização individual de ferramentas de análise e sem recorrer a soluções comerciais.

Com isto em mente, diversas ferramentas SAST *open source* foram analisadas com o intuito de selecionar quatro a integrar na VulnFusion. Estas ferramentas podem ser consultadas na tabela 3.1.

O processo de seleção considerou a popularidade das ferramentas, as funcionalidades disponibilizadas pelas versões gratuitas, o número de tecnologias que conseguem analisar e a aparente facilidade de integração das mesmas. As ferramentas selecionadas foram:

- Semgrep *Community Edition* (CE)
- Spotbugs
- PMD
- Cppcheck

A escolha recaiu sobre ferramentas *open-source* ou com versões comunitárias gratuitas, de modo a garantir acessibilidade, transparência e possibilidade de integração sem custos de licenciamento. Esta abordagem facilita a reprodutibilidade dos

Tabela 3.1: Ferramentas SAST analisadas.

Ferramenta	**Linguagens Suportadas**	**Open-source**	**Gratuito**	**GitHub Stars**
Clang	C, C++, Objective C/C++, OpenCL, CUDA	Sim	Sim	204
Cppcheck	C, C++	Sim	Sim	5800
Frama-C	C	Sim	Sim	167
SpotBugs	Java	Sim	Sim	3500
PMD	Java, JavaScript, Apex, Visualforce, PLSQL, Velocity, XML, XSL	Sim	Sim	4900
Semgrep	30+ linguagens: C#, C, C++, Go, Java, JavaScript, JSON, Python, PHP, Ruby, Scala	Sim	Parcial	10700
SonarQube	Java, C#, JavaScript, Python, PHP, e outras	Sim	Parcial	9100
Infer	Java, C, C++, Objective C	Sim	Sim	15000
Splint	C	Sim	Sim	305
Bandit	Python	Sim	Sim	6500

resultados e a adoção por equipas de desenvolvimento com restrições orçamentais, além de permitir maior flexibilidade na integração e customização da plataforma VulnFusion.

Ferramentas comerciais amplamente reconhecidas, como o Fortify, Checkmarx e as versões comerciais do SonarQube e Semgrep, não foram incluídas devido a custos de licenciamento elevados e ao objetivo de desenvolver uma plataforma aberta e acessível.

De seguida, apresenta-se uma breve descrição das ferramentas selecionadas.

3.1.1 *Semgrep CE*

A Semgrep CE é uma ferramenta de análise estática rápida e *open source*, desenvolvida e mantida pela Semgrep, Inc., que oferece suporte a mais de 30 linguagens, entre as quais C, C++, Java, JavaScript e Python [12].

Esta versão disponibiliza uma vasta gama de regras escritas e mantidas pela comunidade, bem como a possibilidade de desenvolver e utilizar regras personalizadas. A análise está limitada a uma única função ou ficheiro [13], o que pode resultar num maior número de falsos positivos. A análise inclui:

- Propagação de constantes entre funções num único ficheiro – acompanha valores constantes através de várias funções;

- Análise de contaminação (*taint analysis*) numa única função – localiza entradas de informação fornecida por um utilizador para verificar se chegam a operações sensíveis sem passar por um processo de saneamento;
- Análise semântica – proporciona uma melhor compreensão do contexto.

Os resultados da análise podem ser exportados em vários formatos, como JSON, Junit XML, SARIF e texto simples [14].

3.1.2 *Spotbugs com plugin FindSecBugs*

A ferramenta SAST Spotbugs, gratuita e *open source*, foi desenvolvida com o intuito de detetar erros em aplicações Java. O processo de análise é realizado sobre o *bytecode* do projeto alvo e é compatível com Java 11, suporte para versões mais recentes de Java ainda é experimental [15]. A ferramenta utiliza reconhecimento de padrões para encontrar código que possa ser um erro, e disponibiliza mais de 400 padrões de erros em áreas como segurança, vulnerabilidades introduzidas por código malicioso, desempenho, entre outros [16].

É possível expandir as capacidades do Spotbugs com recurso a *plugins*. O *plugin* FindSecBugs adiciona mais de 140 padrões de erros para diferentes tipos de vulnerabilidades como *SQL Injection*, *Cross-Site Scripting* (XSS) e práticas de criptografia insegura [17].

O relatório produzido por esta ferramenta pode ser gerado em formato XML, HTML e SARIF.

Considerando que o Spotbugs analisa apenas *bytecode*, isto requer que o projeto a analisar seja compilado previamente.

3.1.3 *PMD*

PMD é uma ferramenta SAST com suporte para mais de 16 linguagens de programação e com foco particular para Java e Apex. O processo de análise desta ferramenta compara o código fonte com as 400 regras existentes para encontrar possíveis violações [18]. As regras existentes podem ser expandidas com a adição de regras personalizadas.

O relatório do processo de análise pode ser produzido em formatos distintos como XML, JSON, HTML, SARIF entre outros [19].

Apesar do elevado número de regras apresentado por esta ferramenta, o número de regras relativas à segurança é reduzido, o que reduz a eficácia desta ferramenta em análises com foco na segurança do código.

3.1.4 *Cppcheck*

O Cppcheck é uma ferramenta desenvolvida para analisar projetos desenvolvidos em C/C++. O processo de análise é realizado sobre o código fonte, o que torna o processo de compilação do projeto desnecessário.

O processo de análise desta ferramenta tem como objetivo detetar erros no código como por exemplo, comportamentos indefinidos e padrões de código inseguros, e simultaneamente reduzir o número de ocorrências de falsos positivos. O Cppcheck também possui a capacidade de detetar vulnerabilidades de segurança comuns como erros de *buffer* e controlo de acesso inadequado [20].

Os relatórios gerados podem ser exportados nos formatos XML e, em versões mais recentes, em SARIF.

3.1.5 *Comparação dos formatos de relatórios*

Os relatórios produzidos pelas diferentes ferramentas são o ponto de partida para o processo de integração e de onde a informação relativa aos erros é extraída. O formato selecionado deve permitir que a extração da informação desejada seja o mais simples e eficiente possível.

Na Tabela 3.2 encontram-se resumidos os formatos suportados por cada ferramenta, e da qual se pode identificar que apenas dois formatos são comuns a todas as ferramentas, XML e SARIF.

O formato SARIF é um padrão da OASIS, concebido para facilitar a agregação de dados provenientes de múltiplas ferramentas SAST [21].

Por outro lado, o esquema XML é único para cada ferramenta, o que torna a extração de informação mais complexa do que o desejável e exige alterações adicionais sempre que uma nova ferramenta é integrada. As listagens 1 e 2 apresentam excertos dos relatórios em formato XML das ferramentas Semgrep e Spotbugs, respetivamente.

Tabela 3.2: Formato de relatórios por ferramenta.

Formatos	Semgrep CE	Spotbugs	PMD	Cppcheck
XML	X	X	X	X
HTML	-	X	X	-
SARIF	X	X	X	X
JSON	X	-	X	-
Texto	X	-	X	-
Emacs	X	X	X	-

Comparando as listagens, é possível observar que a estrutura e a nomenclatura utilizadas diferem entre ambas. No relatório do Semgrep, a ocorrência identificada encontra-se representada pelo elemento *testcase*, enquanto que no relatório do SpotBugs é utilizada a estrutura *BugInstance*. Além disso, a mesma informação pode ser identificada com designações distintas, por exemplo, a localização do erro no código é descrita através dos atributos *file* e *line* no relatório do Semgrep, mas no SpotBugs é detalhada pelos elementos *sourcefile*, *sourcepath*, *start* e *end*.

Listagem 1: Excerto de relatório XML Semgrep

```
1 <testcase
  ↪ name="java.spring.security.injection.tainted-url-host.tainted-url-host"
  ↪ classname="/src/src/main/java/org/sasanlabs/service/vulnerability/ssrf
2 /SSRFVulnerability.java"
  ↪ file="/src/src/main/java/org/sasanlabs/service/vulnerability/ssrf/
3 SSRFVulnerability.java" line="64">
4 <failure type="ERROR" message="User data flows into the host portion of
  ↪ this manually-constructed URL. This could allow an attacker to send
  ↪ data to their own server, potentially exposing sensitive data such as
  ↪ cookies or authorization information sent with this request. They
  ↪ could also probe internal servers or other resources that the server
  ↪ running this code can access. (This is called server-side request
  ↪ forgery, or SSRF.) Do not allow arbitrary hosts. Instead, create an
  ↪ allowlist for approved hosts, hardcode the correct host, or ensure
  ↪ that the user data can only affect the path or parameters."> URL u =
  ↪ new URL(url);</failure>
5 </testcase>
```

Listagem 2: Excerto de relatório XML Spotbugs

```
<BugInstance type="EI_EXPOSE_REP" priority="2" rank="18" abbrev="EI"
↪  category="MALICIOUS_CODE"
↪  instanceHash="fa28ac89a848a848cc2996456f6d60b1"
↪  instanceOccurrenceNum="0" instanceOccurrenceMax="0" cweid="374">
<ShortMessage>May expose internal representation by returning reference to
↪  mutable object</ShortMessage>
<LongMessage>org.gradle.cli.CommandLineOption.getOptions() may expose
↪  internal representation by returning
↪  CommandLineOption.options</LongMessage>
<Class classname="org.gradle.cli.CommandLineOption" primary="true">
<SourceLine classname="org.gradle.cli.CommandLineOption" start="23"
↪  end="114" sourcefile="CommandLineOption.java"
↪  sourcepath="org/gradle/cli/CommandLineOption.java">
<Message>At CommandLineOption.java:[lines 23-114]</Message>
</SourceLine>
<Message>In class org.gradle.cli.CommandLineOption</Message>
</Class>
<Method classname="org.gradle.cli.CommandLineOption" name="getOptions"
↪  signature="()Ljava/util/Set;" isStatic="false" primary="true">
<SourceLine classname="org.gradle.cli.CommandLineOption" start="37"
↪  end="37" startBytecode="0" endBytecode="46"
↪  sourcefile="CommandLineOption.java"
↪  sourcepath="org/gradle/cli/CommandLineOption.java"/>
<Message>In method org.gradle.cli.CommandLineOption.getOptions()</Message>
</Method>
...
</SourceLine>
</BugInstance>
```

Apesar dos relatórios em XML fornecerem, por vezes, mais detalhes sobre cada erro como por exemplo, o código CWE e excertos do código afetado, a simplicidade de utilização dos relatórios SARIF torna este o formato mais adequado para o trabalho apresentado neste documento.

3.2 ARQUITETURA

A arquitetura de software refere-se à estrutura fundamental de uma aplicação, onde se incluem os componentes principais do sistema e como estes interagem entre si. As escolhas realizadas na definição de uma arquitetura têm impactos significativos no desenvolvimento e manutenção de uma aplicação. Aspetos como a facilidade de adição de novas funcionalidades e escalabilidade de uma aplicação estão diretamente relacionados com a arquitetura escolhida, e esta decisão deve ser cuidada, visto que alterações arquiteturais são frequentemente complexas e morosas [22].

A plataforma VulnFusion, proposta neste documento, tem como objetivo proporcionar uma melhor cobertura na identificação de vulnerabilidades. Esta melhoria é obtida com recurso a diversas ferramentas de análise estática SAST, cujos resultados são submetidos a um processo de normalização e agregação, e posteriormente realiza-se o enriquecimento das vulnerabilidades detetadas com recurso a ferramentas de IA.

Para o desenvolvimento da VulnFusion foram considerados dois aspetos fundamentais, a modularidade e a escalabilidade. A modularidade tem como objetivo facilitar a integração de novas ferramentas SAST e estratégias de enriquecimento baseadas em IA, minimizando possíveis incompatibilidades entre os componentes. A escalabilidade é um aspeto importante para o crescimento da plataforma, onde se equacionam cenários em que a aplicação possa ser alojada num servidor e disponibilizada a múltiplos utilizadores em simultâneo. A arquitetura selecionada para suportar estes requisitos encontra-se representada na figura 3.1, sendo detalhada nos subcapítulos 3.2.1 e 3.2.2.

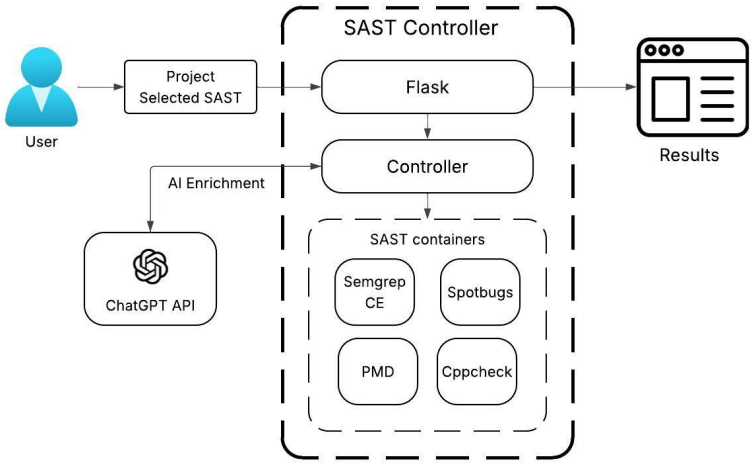


Figura 3.1: Arquitetura da plataforma.

Para assegurar um ambiente de execução consistente, independente do sistema operativo do sistema hóspede, a plataforma foi inserida num contentor *Docker*. O ficheiro *Dockerfile*, indicado na listagem 3, define a imagem base, instala as dependências definidas no ficheiro *requirements.txt* e define os comandos necessários para a sua execução.

Listagem 3: Código Dockerfile.

```
FROM python:3.12-slim

RUN apt-get update && apt-get install -y sudo docker.io gcc g++ libffi-dev

WORKDIR /app

COPY requirements.txt .

RUN pip install --no-cache-dir  -r requirements.txt

CMD ["gunicorn", "-k", "eventlet", "-w", "1", "-b", "0.0.0.0:5000", "main:app"]
```

Adicionalmente, foi utilizado *Docker Compose* para orquestrar os diferentes serviços necessários para a aplicação, permitindo a execução coordenada de múltiplos contentores. Este ficheiro, cujo código pode ser consultado na listagem 4, define três serviços:

- Web: responsável pela execução da aplicação principal, e construido com recurso ao *Dockerfile* mencionado anteriormente.
- Celery: serviço responsável por processar tarefas assíncronas, permitindo a realização do processo de análise e em simultâneo fornecer atualizações em tempo real ao utilizador.
- Redis: utilizado como sistema de armazenamento temporário de dados e comunicação entre os diferentes serviços.

Listagem 4: Código Docker Compose.

```yaml
version: '3.8'

services:
  web:
    build: .
    ports:
      - "5000:5000"
    volumes:
      - ./src:/app
      - /var/run/docker.sock:/var/run/docker.sock
    environment:
      FLASK_ENV: development
      FLASK_DEBUG: 1
      DEBUG: 1
      CELERY_BROKER_URL: redis://redis:6379/0
      CELERY_RESULT_BACKEND: redis://redis:6379/0
      SOCKETIO_MESSAGE_QUEUE: redis://redis:6379/0
    depends_on:
      - redis
      - celery

  celery:
    build: .
    command: celery -A tasks:app worker --loglevel=info
    volumes:
      - ./src:/app
      - /var/run/docker.sock:/var/run/docker.sock
    environment:
      CELERY_BROKER_URL: redis://redis:6379/0
      CELERY_RESULT_BACKEND: redis://redis:6379/0
      SOCKETIO_MESSAGE_QUEUE: redis://redis:6379/0
    depends_on:
      - redis

  redis:
    image: redis:7
    ports:
      - "6379:6379"
```

Esta abordagem facilita a escalabilidade, a manutenção e a replicação do ambiente de desenvolvimento e produção, garantindo que todos os componentes funcionem de forma integrada e isolada do sistema hospedeiro.

Com base na arquitetura acima apresentada, descrevem-se de seguida os componentes principais da aplicação e o seu papel no funcionamento da mesma.

3.2.1 *Interface Web*

A interação do utilizador com a aplicação é realizada através a navegadores de Internet. Esta escolha visa a facilitar o alojamento da aplicação num servidor web.

A aplicação web foi desenvolvida com recurso ao *Flask*, uma *framework* em Python, que permite a criação de interfaces web leves e modulares, facilitando a integração com os restantes componentes da plataforma.

A aplicação disponibiliza três ecrãs principais. O primeiro, ilustrado na figura 3.2, apresenta ao utilizador as funcionalidades de análise disponíveis na VulnFusion. Atualmente, encontram-se disponíveis as seguintes funcionalidades:

- *Project Analyzer*: realiza a análise de um projeto com as ferramentas SAST selecionadas.
- *Project Analyzer with AI*: realiza a análise de um projeto com as ferramentas SAST selecionadas e realiza o enriquecimento dos resultados com recurso a técnicas de IA.
- *File Analyzer with AI*: realiza a análise de um ficheiro de código individual com recurso a técnicas de IA.

Figura 3.2: GUI - ecrã principal.

Selecionando a opção *Project Analyzer with AI*, o utilizador é encaminhado para o ecrã indicado na figura 3.3. Este ecrã disponibiliza três campos de entrada de dados distintos:

- *Project Path*: permite indicar o diretório de projeto a analisar.
- *Auxiliary Path*: permite indicar o diretório que contem ficheiros relevantes para a análise do projeto mas que não se encontram na diretoria do mesmo. Esta opção não é utilizada por todas as ferramentas.
- *Select SAST Tools*: Lista as diferentes ferramentas SAST, das quais o utilizador pode selecionar múltiplas em simultâneo.

O elemento *Supported Languages* lista as linguagens mais relevantes suportadas pelas ferramentas selecionadas.

Após a seleção do projeto e das ferramentas desejadas, o utilizador é encaminhado para o ecrã ilustrado na figura 3.4. Este ecrã fornece ao utilizador atualizações em tempo real relativas ao processo de análise. Uma vez que a análise de um projeto

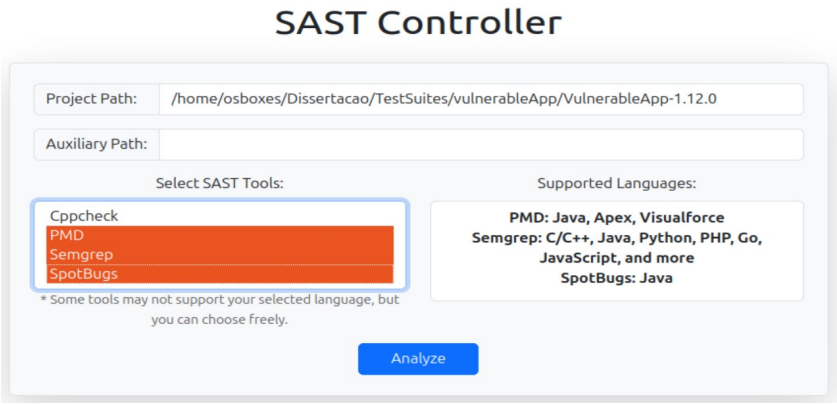


Figura 3.3: GUI - *Project Analyzer with AI* ecrã de seleção.

pode ser um processo moroso, a adição deste ecrã permite assegurar ao utilizador que o processo se encontra a decorrer.

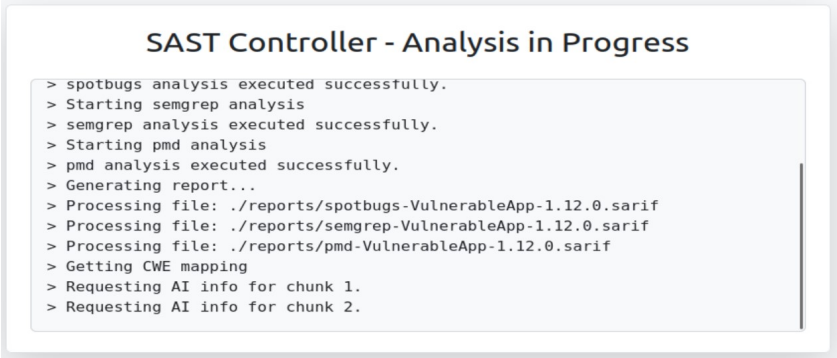
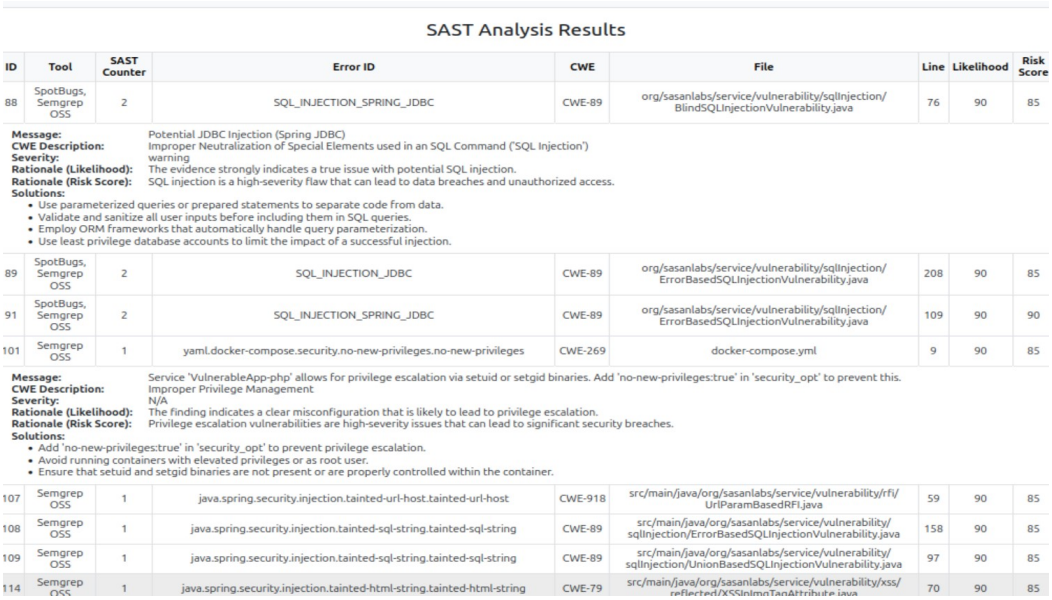


Figura 3.4: GUI - ecrã de progresso.

Com a conclusão do processo de análise, os resultados obtidos são apresentados no ecrã representado na figura 3.5. Os resultados apresentados podem ser ordenados, de forma individual, pelas colunas *id*, *likelihood* e *risk_score*.

De forma a manter a informação apresentada concisa, parte dos resultados associados a cada ocorrência não se encontram diretamente visíveis na tabela. O utilizador pode selecionar a ocorrência desejada, o que permite expandir a mesma e revelar a restante informação. Isto é exemplificado na figura 3.5 pelas ocorrências com os identificadores 88 e 101, onde informação como a descrição da classificação CWE, a fundamentação para a *likelihood* e *risk_score* atribuídos, possíveis soluções de mitigação, entre outras, podem ser consultadas.



SAST Analysis Results

ID	Tool	SAST Counter	Error ID	CWE	File	Line	Likelihood	Risk Score
88	SpotBugs, Semgrep OSS	2	SQL_INJECTION_SPRING_JDBC	CWE-89	org/sasanlabs/service/vulnerability/sqlInjection/BlindSQLInjectionVulnerability.java	76	90	85
	Message: Potential JDBC Injection (Spring JDBC) **CWE Description:** Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') **Severity:** warning **Rationale (Likelihood):** The evidence strongly indicates a true issue with potential SQL injection. **Rationale (Risk Score):** SQL injection is a high-severity flaw that can lead to data breaches and unauthorized access. **Solutions:** • Use parameterized queries or prepared statements to separate code from data. • Validate and sanitize all user inputs before including them in SQL queries. • Employ ORM frameworks that automatically handle query parameterization. • Use least privilege database accounts to limit the impact of a successful injection.							
89	SpotBugs, Semgrep OSS	2	SQL_INJECTION_JDBC	CWE-89	org/sasanlabs/service/vulnerability/sqlInjection/ErrorBasedSQLInjectionVulnerability.java	208	90	85
91	SpotBugs, Semgrep OSS	2	SQL_INJECTION_SPRING_JDBC	CWE-89	org/sasanlabs/service/vulnerability/sqlInjection/ErrorBasedSQLInjectionVulnerability.java	109	90	90
101	Semgrep OSS	1	yaml.docker-compose.security.no-new-privileges.no-new-privileges	CWE-269	docker-compose.yml	9	90	85
	Message: Service 'VulnerableApp-php' allows for privilege escalation via setuid or setgid binaries. Add 'no-new-privileges:true' in 'security_opt' to prevent this. **CWE Description:** Improper Privilege Management **Severity:** N/A **Rationale (Likelihood):** The finding indicates a clear misconfiguration that is likely to lead to privilege escalation. **Rationale (Risk Score):** Privilege escalation vulnerabilities are high-severity issues that can lead to significant security breaches. **Solutions:** • Add 'no-new-privileges:true' in 'security_opt' to prevent privilege escalation. • Avoid running containers with elevated privileges or as root user. • Ensure that setuid and setgid binaries are not present or are properly controlled within the container.							
107	Semgrep OSS	1	java.spring.security.injection.tainted-url-host.tainted-url-host	CWE-918	src/main/java/org/sasanlabs/service/vulnerability/rfi/UrlParamBasedRFI.java	59	90	85
108	Semgrep OSS	1	java.spring.security.injection.tainted-sql-string.tainted-sql-string	CWE-89	src/main/java/org/sasanlabs/service/vulnerability/sqlInjection/ErrorBasedSQLInjectionVulnerability.java	158	90	85
109	Semgrep OSS	1	java.spring.security.injection.tainted-sql-string.tainted-sql-string	CWE-89	src/main/java/org/sasanlabs/service/vulnerability/sqlInjection/UnionBasedSQLInjectionVulnerability.java	97	90	85
114	Semgrep OSS	1	java.spring.security.injection.tainted-html-string.tainted-html-string	CWE-79	src/main/java/org/sasanlabs/service/vulnerability/xss/reflected/XSSInImgTagAttribute.java	70	90	85

Figura 3.5: GUI - ecrã de resultados.

3.2.2 *Controlador*

O controlador é o componente responsável pela execução das tarefas associadas ao processo de análise da aplicação. Este recebe os dados fornecidos pelo utilizador, como o projeto a analisar e as ferramentas SAST selecionadas, e realiza três tarefas principais:

- Lançar as ferramentas SAST para análise
- União dos relatórios produzidos
- Enriquecimento do relatório com recurso a IA

Estas tarefas são realizadas de forma sequencial, sendo necessário a conclusão de cada tarefa para que se possa proceder à inicialização da seguinte. A figura 3.6 apresenta uma representação gráfica desta sequência. Esta abordagem assegura a integridade e consistência dos dados processados.

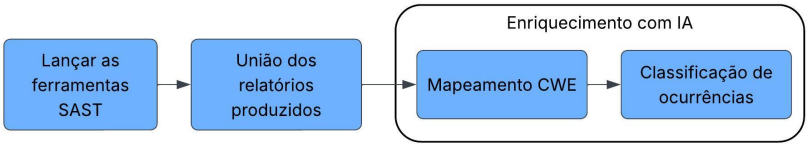


Figura 3.6: Sequência de execução.

3.2.2.1 *Lançar as ferramentas SAST*

As ferramentas SAST são aplicações independentes com requisitos e dependências próprias. A execução de múltiplas ferramentas num único ambiente de execução pode levar à ocorrência de erros devido a incompatibilidades entre as respetivas dependências. Para mitigar este risco, cada ferramenta SAST é executada de forma isolada com recurso a contentores *Docker* dedicados e instanciados quando necessário. Esta abordagem não só facilita a manutenção, atualização ou substituição das ferramentas, como também permite uma gestão mais eficiente dos recursos, dado que os contentores das ferramentas só se encontram em execução quando necessário.

A execução direta de contentores *Docker* a partir de um contentor não é trivial, uma vez que o ambiente interno não possui acesso ao *Docker Daemon*, componente responsável pela gestão de contentores, imagens, volumes e redes. Para contornar esta limitação, foi exposto no serviço web através do ficheiro docker-compose.yml (listagem 4), um volume que referencia o socket do *Docker*, permitindo a comunicação direta com o *daemon* do sistema hospedeiro. A listagem 5 apresenta o volume em questão.

Listagem 5: Excerto Docker Compose referente à socket Docker

```
1       volumes:
2         - ./src:/app
3         - /var/run/docker.sock:/var/run/docker.sock
```

Com exceção da ferramenta Semgrep, que disponibiliza uma imagem Docker oficial, foi necessário desenvolver ficheiros Dockerfile personalizados para as restantes ferramentas utilizadas. Estes ficheiros, apresentados nas listagens 6, 7 e 8, permitem a criação das imagens necessárias para garantir a execução isolada e compatível de cada ferramenta.

O *Dockerfile* da listagem 6 configura o ambiente necessário para a execução da ferramenta SpotBugs, juntamente com o *plugin* FindSecBugs. Na imagem base são instaladas as dependências essenciais, como o JDK, nas linhas 12 a 14 é realizado o download e extração da ferramenta SpotBugs, enquanto que nas linhas 16 a 19 é realizado o mesmo processo para o *plugin* FindSecBugs. O ambiente é configurado para que a ferramenta possa ser executada diretamente via o comando *java -jar*.

Listagem 6: Código Dockerfile Spotbugs

```
1  FROM ubuntu:latest
2
3  RUN apt-get update && apt-get install -y wget unzip openjdk-21-jdk
4  RUN mkdir -p /spotbugs
5
6  WORKDIR /spotbugs
7
8  ENV spotbugs_version="4.9.1"
9  ENV findsec_version="1.12.0"
10
11 RUN set -o errexit && \
12     wget -nc --quiet "https://github.com/spotbugs/spotbugs/releases/download/${s
       ↪ potbugs_version}/spotbugs-${spotbugs_version}.zip" && \
13     unzip -o "spotbugs-${spotbugs_version}.zip" && \
14     rm "spotbugs-${spotbugs_version}.zip" && \
15     cd "spotbugs-${spotbugs_version}"/plugin && \
16     wget -nc --quiet
       ↪ "https://github.com/find-sec-bugs/find-sec-bugs/releases/download/versio
       ↪ n-${findsec_version}/findsecbugs-cli-${findsec_version}.zip" && \
17     unzip -o "findsecbugs-cli-${findsec_version}.zip"
       ↪ "lib/findsecbugs-plugin-${findsec_version}.jar" && \
18     mv "lib/findsecbugs-plugin-${findsec_version}.jar" . && \
19     rm "findsecbugs-cli-${findsec_version}.zip" && \
20     rmdir lib
21
22 ENV PATH="/spotbugs/spotbugs-${spotbugs_version}/lib:${PATH}"
23 ENTRYPOINT ["java", "-jar","/spotbugs/spotbugs-4.9.1/lib/spotbugs.jar"]
```

O *Dockerfile* da listagem 7 prepara o ambiente para a execução da ferramenta PMD. Tal como no exemplo anterior, são instaladas as dependências necessárias, como o JDK, nas linhas 9 a 10 realiza-se o download e extração da distribuição binária da versão especificada do PMD. O PATH é configurado para permitir a execução direta da ferramenta através do comando *pmd*.

Listagem 7: Código Dockerfile PMD

```
1  FROM ubuntu:latest
2
3  RUN apt-get update && apt-get install -y wget unzip openjdk-21-jdk
4
5  WORKDIR /pmd
6
7  ENV pmd_version="7.10.0"
8
9  RUN wget -nc "https://github.com/pmd/pmd/releases/download/pmd_releases/${pmd_ve
   ↪ rsion}/pmd-dist-${pmd_version}-bin.zip" && \
10     unzip "pmd-dist-${pmd_version}-bin.zip" && \
11     rm -f "pmd-dist-${pmd_version}-bin.zip"
12
13 ENV PATH="/pmd/pmd-bin-${pmd_version}/bin:${PATH}"
14 ENTRYPOINT ["pmd"]
```

O *Dockerfile* da listagem 8 configura o ambiente para a ferramenta Cppcheck. Na imagem base são instaladas ferramentas de compilação, como *make* e *g++*, e

posteriormente, linhas 10 a 14, o código-fonte da versão desejada do Cppcheck é descarregado e compilado. A execução da ferramenta realiza-se através do comando *cppcheck*.

Listagem 8: Código Dockerfile Cppcheck

```
1  FROM ubuntu:latest
2
3  RUN apt-get update && apt-get install -y wget unzip make g++
4  RUN mkdir -p /cppcheck
5
6  WORKDIR /cppcheck
7
8  ENV cppcheck_version="2.16.0"
9
10 RUN  wget -nc "https://github.com/danmar/cppcheck/archive/refs/tags/${cppcheck_v
   ↪  ersion}.zip" && \
11     unzip "${cppcheck_version}.zip" && \
12     rm -f "${cppcheck_version}.zip" && \
13     cd "cppcheck-${cppcheck_version}" && \
14     make
15
16 ENV PATH="/cppcheck/cppcheck-${cppcheck_version}:${PATH}"
17 ENTRYPOINT ["cppcheck"]
```

As ferramentas SAST são instanciadas através de comandos padronizados, definidos para garantir uma execução consistente e eficiente. Estes comandos asseguram uma cobertura equilibrada entre o nível de detalhe da análise e a sua duração. A listagem 9 apresenta o método utilizado para instanciar a ferramenta SpotBugs, como exemplo representativo da abordagem adotada para as restantes ferramentas. É possível observar, nas linhas 2 a 5, a definição dos volumes a montar no contentor *docker* da ferramenta, e na linha 6 a definição do comando a executar pela ferramenta.

Embora algumas ferramentas permitam a utilização de regras personalizadas, optou-se por utilizar apenas os conjuntos de regras pré-definidos. Esta decisão deve-se ao facto de a elaboração de novas regras não se enquadrar no âmbito do trabalho descrito neste documento, além de que muitas das regras personalizadas disponíveis estão associadas às versões comerciais das ferramentas.

Listagem 9: Código do método que instancia Spotbugs

```
def spotbugs(project_name, project_dir, reports_dir):
    volumes = {
        project_dir: {'bind': '/opt', 'mode': 'ro'},
        reports_dir: {'bind': '/reports', 'mode': 'rw'}
    }
    command = f"-textui -effort:more -medium -progress -pluginList
    ↪ findsecbugs-plugin-1.12.0.jar
    ↪ -sarif=/reports/spotbugs-{project_name}.sarif /opt"
    client = docker.DockerClient(base_url='unix://var/run/docker.sock')
    container = client.containers.run(
        "spotbugs",
        command=command,
        volumes=volumes,
        detach=True,
        tty=True
    )
    result = container.wait()
    exit_code = result['StatusCode']
    container.remove()
    return exit_code
```

O resultado final desta etapa consiste num ficheiro SARIF gerado por cada ferramenta instanciada, e armazenado na diretoria da plataforma. Estes ficheiros contêm os registos de todas as ocorrências detetadas durante a análise, servindo como base para as etapas subsequentes de consolidação e enriquecimento dos resultados.

3.2.2.2 *União dos relatórios*

A análise de um projeto com recurso a múltiplas ferramentas SAST pode resultar na identificação repetida da mesma ocorrência por diferentes ferramentas. A apresentação de um relatório final com múltiplas entradas para a mesma ocorrência torna a sua análise mais complexa e pode ter um impacto negativo nos esforços de mitigação.

Para simplificar a análise e interpretação dos resultados, foi desenvolvido um algoritmo, apresentado na figura 3.7, preservando simultaneamente a informação sobre quais as ferramentas que as identificaram.

O processo de união tem como objetivo identificar e agrupar ocorrências que correspondam ao mesmo erro lógico, mesmo quando são identificadas de forma distinta pelas diferentes ferramentas SAST. Para realizar esta comparação, são considerados dois elementos principais, o identificador do erro (*error_id*), específico a cada ferramenta, e a localização do erro, que inclui o nome do ficheiro e a linha de código onde a ocorrência foi detetada.

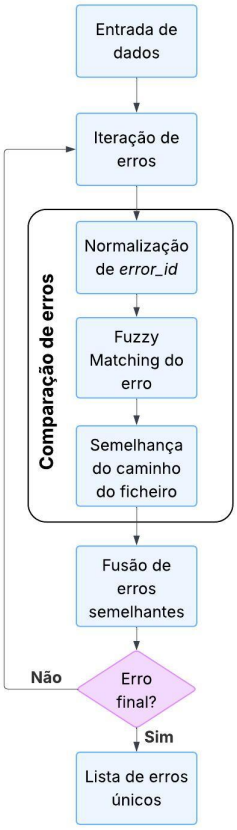


Figura 3.7: Algoritmo para união dos relatórios.

Como mencionado anteriormente, cada ferramenta possui nomenclaturas distintas para os identificadores dos erros. Este facto inviabiliza a utilização de comparação direta para averiguar se ocorrências provenientes de ferramentas distintas correspondem ao mesmo erro lógico. Para ultrapassar esta limitação, os identificadores são submetidos a um processo de normalização, seguido de uma comparação com recurso a técnicas de *fuzzy matching*. O processo de normalização, apresentado na listagem 10, tem como objetivo transformar os identificadores de forma a que possam ser comparáveis. Para isto realizam-se as seguintes transformações:

- Substituição de pontuação por espaços.
- Separação de *Camel Case*.
- Transformar os caracteres em minúsculas.
- Eliminar espaços sequenciais.

Estas transformações permitem reduzir variações superficiais entre identificadores que representam o mesmo conceito. Após a normalização, os identificadores são convertidos em *tokens* e comparados utilizando o método *token_set_ratio*, da

biblioteca `rapidfuzz` [23]. Este método calcula uma métrica de similaridade com base na proporção de tokens coincidentes entre duas cadeias de texto, ignorando a ordem e a duplicação dos mesmos, o que o torna especialmente eficaz na deteção de equivalências semânticas entre identificadores com estruturas distintas.

Listagem 10: Normalização de error_id.

```
def normalize_error_id(error_id: str) -> str:
    # Step 1: Replace punctuation (., -, _) with space
    normalized = re.sub(r'[-_.]', ' ', error_id)

    # Step 2: Split camel case
    normalized = re.sub(r'([a-z])([A-Z])', r'\1 \2', normalized)

    # Step 3: Lowercase everything
    normalized = normalized.lower()

    # Step 4: Collapse multiple spaces into one
    normalized = re.sub(r'\s+', ' ', normalized).strip()

    return normalized
```

A comparação entre ficheiros é realizada pelo método apresentado na listagem 11, que avalia o nome e o caminho dos ficheiros onde os erros foram detetados. Na linha 5, é realizada a verificação inicial para garantir que os nomes dos ficheiros coincidem, e nas linhas 8 a 11, é avaliado o número de partes comuns no caminho dos ficheiros, começando a comparação a partir do final da estrutura de diretórios. Esta abordagem permite ultrapassar variações introduzidas pelas ferramentas SAST, como prefixos ou diretórios temporários adicionados durante o processo de análise.

Além da comparação de caminhos, é também considerada a linha de código onde o erro foi detetado, com uma margem de tolerância de ± 5 linhas. Ferramentas SAST distintas podem apontar para linhas ligeiramente diferentes dentro do mesmo bloco de código vulnerável, devido a variações nos seus mecanismos de análise. A adição desta tolerância permite ter em conta estas pequenas variações, mantendo a capacidade de identificar ocorrências que correspondem ao mesmo erro lógico.

Listagem 11: Semelhança de ficheiros.

```
def path_similarity(path1, path2):
    p1, p2 = Path(path1), Path(path2)

    # Compare file names directly
    if p1.name != p2.name:
        return 0

    # Count shared parts from the end (package structure)
    p1_parts = p1.parts[::-1]
    p2_parts = p2.parts[::-1]
    match_len = sum(a == b for a, b in zip(p1_parts, p2_parts))

    # Normalize score between 0 and 100
    smallest_path_len = min(len(p1.parts), len(p2.parts))
    return int((match_len / smallest_path_len) * 100)
```

Ambos os processos previamente descritos resultam na produção de um valor numérico numa escala 0 a 100. Definiu-se um limiar de similaridade de 75 para ambos os critérios, considerando-se que valores iguais ou superiores apresentam uma correspondência suficientemente forte para que duas ocorrências sejam consideradas equivalentes.

O limiar de similaridade de 75 foi definido como um ponto de partida lógico, com base na necessidade de equilibrar a sensibilidade da comparação. Este valor procura evitar uma correspondência demasiado restritiva, que poderia ignorar ocorrências equivalentes, e ao mesmo tempo prevenir uma correspondência demasiado permissiva, que poderia agrupar erros distintos. Embora não tenha sido estabelecido com base em dados empíricos, este valor foi considerado adequado para os objetivos iniciais do algoritmo.

A lógica de decisão baseia-se na verificação conjunta dos critérios de similaridade e da proximidade da linha de código. A listagem 12, demonstra a estrutura de decisão que considera duas ocorrências idênticas quando ambos os critérios de similaridade atingem ou ultrapassam o limiar definido, e a linha de código se encontra dentro da margem de tolerância estabelecida.

Listagem 12: Estrutura de decisão de similaridade

```
if (score_error > 75 and score_location > 75 and
    row['line'] - 5 <= merged_row['line'] <= row['line'] + 5):

    # Considera-se que as ocorrências são equivalentes
```

Esta conclusão desta etapa produz dois artefactos distintos, necessários para o processo de enriquecimento. Estes artefactos são o resultado da união das ocorrências e uma lista de identificadores de erro únicos.

3.2.2.3 *Enriquecimento do relatório com recurso a IA*

A presente etapa tem como objetivo enriquecer os relatórios produzidos pelas ferramentas SAST através da aplicação de técnicas de inteligência artificial. Pretende-se fornecer ao utilizador informações adicionais e mais detalhadas sobre os erros identificados, com o intuito de apoiar a sua prioritização e a mitigação eficaz das vulnerabilidades detetadas.

Para este fim, optou-se pela utilização da API da OpenAI, uma solução amplamente disponível e acessível. Esta escolha foi motivada pela facilidade de integração, pelas capacidades dos modelos disponíveis e pela possibilidade de qualquer utilizador replicar o processo, apesar dos custos associados ao uso da API. A opção de utilizar uma solução externa permitiu focar os esforços no desenvolvimento da plataforma VulnFusion, evitando o investimento significativo de tempo e recursos necessários para o desenvolvimento e treino de modelos personalizados.

O processo de enriquecimento é composto por duas tarefas distintas. A primeira visa a realizar o mapeamento entre as ocorrências identificadas para respetiva categoria CWE. A segunda tarefa consiste na estimativa da probabilidade de cada ocorrência ser um verdadeiro positivo, e na avaliação de risco associado.

Os pedidos realizados à API da OpenAI para obter esta informação são efetuados em blocos, de forma a respeitar os limites impostos pelos modelos utilizados. Esta abordagem permite otimizar o desempenho, ajustando a dimensão dos blocos, e garantir a conformidade com as restrições de utilização da API.

Mapeamento CWE

Como referido anteriormente, os relatórios SARIF produzidos pelas ferramentas SAST apresentam, em regra geral, apenas o identificador do erro e a respetiva mensagem para cada ocorrência detetada. Embora esta informação seja útil, não inclui elementos frequentemente disponibilizados por outros formatos de relatório, como a categoria CWE e o bloco de código onde a ocorrência foi identificada. Esta tarefa procura colmatar essa lacuna, enriquecendo os dados com a classificação CWE e com recomendações práticas de mitigação.

Com base nos testes abordados no capítulo seguinte, optou-se por implementar um processo de mapeamento com duas fases de raciocínio distintas, obtidas pela realização de dois pedidos distintos à API da OpenAI. Os dados utilizados neste processo são gerados pela etapa anterior e consistem numa lista de identificadores de erros únicos, cada um acompanhado pela respetiva mensagem. Estes dados são agrupados em blocos de erros, o que permite respeitar os limites da API e otimizar o desempenho, ajustando a dimensão dos mesmos.

O primeiro pedido inclui o identificador do erro e a respetiva mensagem, com o objetivo de obter a categoria CWE correspondente, a sua descrição oficial e recomendações práticas para mitigar a ocorrência. O *prompt* utilizado é composto por duas instruções distintas:

- *system* define as instruções de alto nível, como o contexto e regras de resposta (listagem 13).
- *user* fornece os dados e algumas instruções adicionais, como por exemplo o formato da resposta (listagem 14).

Listagem 13: Mapeamento CWE - primeiro pedido instrução *system*

```
You are an expert in application security and vulnerability analysis with
deep knowledge of MITRE CWE classifications.
For each SAST error, identify the most accurate CWE identifier and
its official description,and recommend 1-2 concise, industry-standard
solutions.
### RESPONSE RULES:
- Return **only** a valid JSON array of objects.
- Use **double quotes** for all keys and string values.
- No markdown, headings, or extraneous text.
- No semicolons (`;`). No trailing commas.
- Maintain consistent mappings for duplicate errors.
```

Listagem 14: Mapeamento CWE - primeiro pedido instrução *user*

```
### Error List:
{k}: {v} para cada k,v em chunk.items()

### Output format:
Return exactly a JSON array where each object has these keys, in this
↪  order:
  sast_error, cwe, cwe_description, solutions
- `sast_error`: the error ID string
- `cwe`: official CWE identifier (e.g., "CWE-79")
- `cwe_description`: official CWE title
- `solutions`: array of 1-2 solution strings
No extra keys or text.
```

A introdução da segunda fase de raciocínio pretende tornar o processo de mapeamento CWE mais robusto e fiável, através de uma reavaliação crítica dos dados iniciais e dos resultados obtidos no pedido anterior, considerados em simultâneo. Tal como na primeira fase, o *prompt* utilizado neste pedido é composto por duas instruções, contudo, apenas se apresenta a instrução *system* (listagem 15), uma vez que a instrução *user* se limita à inserção dos dados a analisar, sem conter instruções adicionais relevantes.

Listagem 15: Mapeamento CWE - excerto do segundo pedido instrução *system*.

```
You are a senior security analyst with deep expertise in application security and the MITRE CWE taxonomy.
For each SAST finding below, re-evaluate the initial CWE mapping in context of the raw error text,
and then output only a JSON array of objects, where each object has exactly these keys in this order:
  1. sast_error
  2. verified
  3. cwe
  4. cwe_description
  5. justification
  6. solutions
  7. reasoning

Rules for the JSON:
- Use double quotes only; no single quotes, no semicolons, no markdown, no extra fields.
- `verified`: true if the initial mapping is correct; false otherwise.

If `verified` is true:
- Keep `cwe` and `cwe_description` exactly as provided.
- You MAY leave `justification` as `""`.
- **Re-emit** the original `solutions` array unchanged.
If `verified` is false:
1. **Replace** `cwe` with a different, correct CWE identifier (it must not equal the initial value).
2. **Replace** `cwe_description` with that CWE's official title.
3. Provide a concise `justification` (1-2 sentences) explaining why the original mapping was incorrect.
4. Provide at least 2 *brand-new* `solutions` that you have *not* mentioned in the first stage.
```

Esta segunda análise permite validar ou corrigir o mapeamento inicial, justificando cada decisão com base no conteúdo do erro e na taxonomia oficial CWE. O modelo deve fornecer, no atributo *reasoning*, o processo de decisão envolvido na análise dos dados submetidos, detalhando o raciocínio que suporta cada mapeamento. No caso de se verificar a alteração o CWE originalmente atribuído, deve ser apresentado um novo identificador CWE, acompanhado da sua descrição oficial, duas novas soluções

específicas para a nova categoria e uma breve justificação que explique a correção efetuada.

A informação obtida neste processo de mapeamento é integrada na união das ocorrências, permitindo a sua utilização na etapa seguinte. É relevante salientar que nem todos os campos gerados nesta etapa são fornecidos ao utilizador na camada de apresentação, sendo alguns reservados exclusivamente para registo interno, processos de depuração e melhoria contínua dos *prompts* utilizados.

Classificação de ocorrências.

A segunda tarefa na etapa de enriquecimento tem como objetivo atribuir a cada ocorrência identificada uma estimativa da sua veracidade e do risco de segurança que esta representa. Esta informação é essencial para apoiar os utilizadores na prioritização dos esforços de mitigação, permitindo que as vulnerabilidades mais relevantes sejam abordadas em primeiro lugar.

A lista de ocorrências, previamente enriquecida com a classificação CWE, é enviada à API da OpenAI com o *prompt* apresentado na listagem 16. Este *prompt* define a estrutura dos dados a analisar, os objetivos e condições do processo de análise e o formato esperado da resposta.

Listagem 16: Prompt para a classificação de erro

```
You are a security triage assistant.

You will receive a JSON array of static application security testing (SAST) findings. Each item includes:
- id
- tool
- error_id
- cwe (optional)
- cwe_description (optional)
- solutions (optional)
- severity (optional)
- msg
- location_file
- line
- sast_counter

Your task is to analyze each item and return the same list, **adding four new fields** to each object:
1. likelihood (integer 0-100): how likely this finding is a true positive
2. rationale_likelihood (string): brief justification for the likelihood score
3. risk_score (integer 0-100): how severe or security-significant this finding is
4. rationale_risk_score (string): brief justification for the risk_score

=== CRITICAL INSTRUCTIONS ===
1. Return the **exact same number of items** as received. No filtering or removal.
2. Use the `id` to match inputs to outputs one-to-one.
3. Keep all original fields intact and unchanged.
4. Response must be a **raw JSON array only** - no markdown, labels, comments, or explanations.
5. Do not include the character ";" in the response.
6. Order of the list must be the same as received.

Guidance for scoring:
- **likelihood**:
  • 0-30 if the tool often misfires on this pattern
  • 31-70 if there's some ambiguity
  • 71-100 if the evidence strongly indicates a true issue
- **risk_score**:
  • 0-30 for naming/style or low-impact issues
  • 31-70 for moderate findings (e.g. weak encryption, info exposure)
  • 71-100 for high-severity flaws (e.g. SQLi, RCE, privilege escalation)

CRITICAL: Do NOT remove or merge any items.
If you cannot process an item for any reason, output it with:
  "likelihood": 50,
  "rationale_likelihood": "insufficient information to adjust from midpoint",
  "risk_score": 50,
  "rationale_risk_score": "insufficient information to adjust from midpoint"
```

Após a avaliação dos erros, o modelo deve produzir os seguintes campos:

- *likelihood*: apresenta um valor entre 0 e 100 que representa a probabilidade do erro ser um verdadeiro positivo;
- *rationale_likelihood*: contem uma breve descrição para o valor atribuído à *likelihood*;
- *risk_score*: apresenta um valor entre 0 e 100 que reflete a gravidade que o erro apresenta para a segurança do projeto;
- *rationale_risk_score*; breve justificação para o valor atribuído ao *risk_score*.

A existência de duas propriedades distintas que permitem, de forma individual, orientar as decisões dos utilizadores, a veracidade da ocorrência e o risco de segurança associado, possibilita a criação de uma métrica composta que conjugue ambas as dimensões. Para tal, procedeu-se à normalização do *risk_score*, apresentado na tabela 3.3, com o intuito de obter uma escala qualitativa que facilite a interpretação e comparação entre ocorrências. A normalização foi realizada com base na seguinte fórmula:

$$\text{Fator de Risco} = 1 + \left(\frac{\text{risk\_score}}{100}\right) \times 9$$

Tabela 3.3: Classificação de risco por nível de severidade.

Fator de Risco [1–10]	**Rótulo**	**Descrição**	**Mapeamento [*risk_score*]**
1 - 2	Insignificante	Sem perda de dados ou impacto no serviço.	**0 - 17**
3 - 4	Baixo	Exposição menor de dados, utilizadores limitados, sem interrupção de serviço.	**18 - 39**
5 - 6	Moderado	Exposição significativa de dados ou degradação limitada do serviço.	**40 - 61**
7 - 8	Elevado	Perda significativa de dados, bypass de autenticação, interrupção relevante.	**62 - 83**
9 - 10	Crítico	Execução remota de código, comprometimento total do sistema.	**84 - 100**

A métrica composta, designada por *Security and Quality Impact* (SQI), é calculada com base na probabilidade de veracidade da ocorrência (*likelihood*, L, expressa em percentagem), no fator de risco normalizado (R), e em dois pesos ajustáveis (α e β) que permitem calibrar a influência relativa de cada componente. A fórmula utilizada é apresentada em seguida:

$$\text{SQI} = \alpha L \times \beta R$$

O sistema inclui um mecanismo de verificação de integridade que compara o número de ocorrências obtidos na resposta coincide com o número de ocorrências enviadas no pedido. Caso contrário, é emitida uma exceção e o pedido é repetido automaticamente. Este controlo é fundamental para garantir que o processo de enriquecimento não introduz inconsistências no relatório final.

Esta tarefa utiliza o modelo *gpt-4o-mini* através da API *chat completions*. Embora seja tecnicamente possível fornecer múltiplas ocorrências ao modelo num único pedido, dentro dos limites definidos na documentação, verificou-se que essa abordagem aumenta a probabilidade de discrepâncias entre as ocorrências fornecidas e as devolvidas. Esta limitação pode ser mitigada com a utilização de modelos mais avançados, como o *gpt-4.1-mini*, embora isso implique um aumento no custo por pedido.

3.3 SÍNTESE

Este capítulo descreve o desenvolvimento da plataforma VulnFusion. Realiza-se a descrição e análise das ferramentas SAST *open-source* selecionadas, e apresenta-se de forma detalhada a arquitetura modular da plataforma e os processos de integração, normalização e enriquecimento dos resultados com recurso a inteligência artificial.

O Capítulo 4 apresenta os testes realizados para validar a eficácia da plataforma VulnFusion, incluindo a metodologia de avaliação, os resultados obtidos em diferentes categorias de vulnerabilidades e a análise comparativa com estudos de referência.

4

RESULTADOS

O trabalho desenvolvido na etapa de **Enriquecimento do relatório com recurso a IA** requer validação de forma a determinar a sua eficácia. Este capítulo tem como objetivo descrever a metodologia de testes implementada para os processos de **Mapeamento CWE** e **Classificação de Ocorrências**, e apresentar os resultados obtidos.

4.1 MAPEAMENTO CWE

O processo de mapeamento de CWE realizado com recurso à API da OpenAI pode apresentar alguns problemas inerentes aos próprios modelos, como por exemplo não analisar toda a informação fornecida e interpretar de forma ambígua os *prompts* fornecidos.

Com o intuito de avaliar a consistência da informação obtida neste processo, foram executados um conjunto de testes tendo como base a aplicação VulnerableApp [24] desenvolvida por Sansanlabs. Esta aplicação foi concebida com vulnerabilidades de forma a servir de plataforma de aprendizagem e testes na área de cibersegurança. A análise desta aplicação com as ferramentas SAST selecionadas e a união dos respetivos relatórios produziu uma lista de 62 identificadores de erro únicos com as respetivas mensagens associadas.

A metodologia dos testes consiste em executar o processo de mapeamento de CWE trinta vezes com a mesma lista de erros. Os resultados obtidos são resumidos através das seguintes métricas:

- Média de valores "N/A"por execução: percentagem média de quantos dos 62 identificadores de erro foram efetivamente mapeados para um identificador CWE.
- Média da dominância por erro: proporção média do valor mais frequente por erro, em que 100% significa que, em todas as tentativas, o mesmo erro foi mapeado consistentemente para o mesmo identificador CWE.

Em suma, o objetivo destes testes não é validar se o mapeamento dos erros é correto, mas sim medir a capacidade do processo de mapeamento de identificar o maior número possível de CWEs, e avaliar a consistência desses resultados.

Algumas das APIs utilizadas na realização destes testes e implementadas na VulnFusion disponibilizam um parâmetro designado por *temperatura*, que controla o grau de aleatoriedade nas respostas geradas pelos modelos [25]. Valores baixos, próximos de 0, tornam as respostas mais determinísticas e consistentes, enquanto valores mais elevados introduzem maior variabilidade e criatividade nas respostas. Para garantir consistência nos testes realizados, foi utilizada, sempre que possível, uma temperatura igual a zero, com o intuito de que o modelo produzisse sempre a mesma resposta para o mesmo conjunto de dados.

Devido à extensão dos *prompts* utilizados durante o processo de desenvolvimento e de testes, estes encontram-se inseridos nos anexos sendo apenas referenciados nos respetivos testes.

4.1.1 *Pedido único de mapeamento*

A presente secção apresenta os testes realizados onde apenas um pedido de mapeamento é feito à API.

4.1.1.1 *Teste 1*

Este teste utiliza exclusivamente o identificador de erro fornecido pelas ferramentas SAST. São comparados três *prompts* distintos e o pedido de mapeamento é realizado em blocos de 15 erros. O modelo utilizado neste teste foi o "gpt-4.1-mini"com o parâmetro de temperatura a zero, de modo a tornar as respostas mais determinísticas.

Os *prompts* utilizados na realização deste teste podem ser consultados no anexo A.1.

Tabela 4.1: Mapeamento CWE - teste 1.

Prompt	**Média de não "N/A" por tentativa(%)**	**Média de dominância por erro(%)**
Prompt 1	48,17	95,16
Prompt 2	58,28	88,28
Prompt 3	59,03	85,38

O conteúdo da tabela 4.1 indica que o *prompt* 1 produz resultados mais consistentes (maior dominância média), embora apresente a cobertura mais baixa. Na generalidade, todos os *prompts* apresentam um valor de cobertura baixo.

4.1.1.2 *Teste 2*

O Teste 2 procura avaliar se a inclusão da mensagem de erro contribui para melhorar os resultados obtidos no teste anterior. Para isso, foram adaptados os três *prompts* do Teste 1, de forma incluir como input adicional o texto da mensagem de erro. A dimensão dos blocos, o modelo e a sua configuração foram mantidos. Estes *prompts* podem ser consultados no anexo A.2.

Tabela 4.2: Mapeamento CWE - teste 2.

Prompt	**Média de não "N/A" por tentativa(%)**	**Média de dominância por erro(%)**
Prompt 1	57,53	95,59
Prompt 2	58,01	97,42
Prompt 3	61,13	93,87

Os resultados obtidos, indicados na tabela 4.2, indicam que a adição da mensagem do erro levou a:

- Melhoria marginal na cobertura de mapeamento.
- Aumento significativo na consistência do mapeamento para os *prompts* 2 e 3.

É possível afirmar que a adição da mensagem de erro torna o modelo mais eficaz na atribuição de identificadores CWE.

4.1.1.3 *Teste 3*

Devido à natureza dos modelos de inteligência artificial, o envio de blocos com um elevado número de erros pode produzir resultados inesperados mesmo com *prompts* afinados para minimizar este efeito. O teste 3 reutiliza os *prompts* do teste anterior, mas reduz a dimensão do bloco a um erro por pedido. O modelo utilizado foi o "gpt-4.1-mini"com a temperatura a zero.

Comparativamente com o teste anterior, a limitação da dimensão do bloco a um erro, representada na tabela 4.3, produz uma melhoria significativa na cobertura do

Tabela 4.3: Mapeamento CWE - teste 3.

Prompt	**Média de não "N/A" por tentativa(%)**	**Média de dominância por erro(%)**
Prompt 1	84,95	90,78
Prompt 2	85,81	90,11
Prompt 3	88,49	87,90

mapeamento, ultrapassando os 80% nos três *prompts*. Contudo, verifica-se também uma redução na consistência dos resultados.

Em suma, o envio individual de cada erro potencia a capacidade do modelo para encontrar um CWE correspondente, mas introduz alguma inconsistência nos resultados.

4.1.2 *Pedido inicial e confirmação de mapeamento*

Os testes apresentados nesta secção realizam dois ou mais pedidos distintos por bloco de erros, com o intuito de averiguar se um processo de confirmação sobre o mapeamento inicial tem um impacto significativo nos resultados obtidos.

4.1.2.1 *Teste 4*

O teste 4 implementa um processo de mapeamento em duas etapas distintas. A primeira etapa fornece o identificador de erro e a mensagem para obter a categoria CWE. Na segunda etapa, os dados obtidos do primeiro pedido são fornecidos com o intuito de realizar um processo de verificação. Mantém-se o modelo gpt-4.1-mini com temperatura zero, para garantir respostas determinísticas.

Os *prompts* utilizados na realização deste teste podem ser consultados no anexo A.3.

Tabela 4.4: Mapeamento CWE - teste 4.

	Média de não "N/A" por tentativa(%)	**Média de dominância por erro(%)**
Prompt Teste 4	87,15	85,91

Os resultados obtidos, indicados na tabela 4.4, apresentam valores semelhantes aos apresentados no teste 3. Contudo a realização de dois pedidos distintos por bloco de erros apresenta custos e latência adicionais.

4.1.2.2 *Teste 5*

Com base na ideia de que a adição de uma terceira etapa de verificação poderia apresentar benefícios na consistência dos resultados, o teste 5 expande o teste anterior adicionando um novo pedido de verificação, utilizando o mesmo modelo e parâmetro de temperatura.

A totalidade dos *prompts* utilizados neste teste encontram-se listados no anexo A.4.

Tabela 4.5: Mapeamento CWE - teste 5.

	Média de não "N/A" por tentativa(%)	**Média de dominância por erro(%)**
Prompts Teste 5	88.23	85,05

Em comparação com os resultados do teste 4, o terceiro pedido aumenta ligeiramente a cobertura de mapeamento, mas apresenta um decréscimo marginal na consistência. No entanto, aumenta os custos e latência do processo.

4.1.2.3 *Teste 6*

Enquanto que os *prompts* utilizados nos teste 4 e 5 implementam processos de verificação do mapeamento original, no teste 6 tenciona-se submeter os dados obtidos do mapeamento original a um novo processo de análise crítica. Para isso, desenvolveram-se novos *prompts*, onde na segunda etapa se fornece simultaneamente os dados obtidos do mapeamento inicial com o identificador e a mensagem do erro. Os resultados podem ser observados na tabela 4.6.

Os *prompts* que originaram os seguintes resultados podem ser observados no apêndice A.5.

É possível observar uma subida substancial na cobertura do mapeamento, mantendo um valor de consistência de resultados semelhante aos obtidos nos testes 4 e 5. Estes ganhos justificam o acréscimo de custos associados.

Tabela 4.6: Mapeamento CWE - teste 6.

	Média de não "N/A" por tentativa(%)	**Média de dominância por erro(%)**
Prompts Teste 6	99.57	86.56

4.1.2.4 *Teste 7*

O presente teste adiciona ao teste anterior, um terceiro processo de análise cujo *prompt* recebe o identificador do erro e mensagem, e os dados obtidos das etapas anteriores. Os resultados obtidos podem ser observados na tabela 4.7.

Os *prompts* referentes a este teste podem ser consultados no anexo A.6.

Tabela 4.7: Mapeamento CWE - teste 7.

	Média de não "N/A" por tentativa(%)	**Média de dominância por erro(%)**
Prompts Teste 7	99.41	83.12

Em relação ao teste 6 é possível observar um valor idêntico de cobertura e uma ligeira redução da consistência dos resultados.

4.1.2.5 *Teste 8*

Os testes realizados anteriormente utilizam um modelo, "gpt-4.1-mini", que não permite a preservação do contexto e do raciocínio realizado pelo modelo entre pedidos distintos.

Este teste visa a averiguar se a preservação deste contexto entre pedidos influencia positivamente os resultados. Utilizou-se o modelo "o4-mini"com a API *responses*, que apresenta custos de utilização mais elevados, mas permite ligar os dois pedidos passando o identificador entre os dois pedidos.

Os *prompts* desenvolvidos podem ser consultados no anexo A.7.

Comparativamente com os resultados obtidos nos testes anterior, a tabela 4.8 mostra uma redução na cobertura e na consistência do mapeamento. Associado ao acréscimo de custos, torna esta opção pouco viável.

Tabela 4.8: Mapeamento CWE - teste 8.

	Média de não "N/A" por tentativa(%)	**Média de dominância por erro(%)**
Prompts Teste 8	64,35	81,08

4.1.3 *Resultados*

Observando a totalidade dos resultados obtidos nos testes anteriores, os testes onde o processo de mapeamento é realizado num único pedido apresenta valores médios de dominância por erro superiores, de onde se pode destacar o teste 3 que apresenta também valores de cobertura média elevados. Dos testes onde o mapeamento é realizado em múltiplas etapas, é possível destacar o teste 6 que apresenta uma cobertura perto dos 100% e em simultâneo um valor médio de consistência de resultados elevado.

Considerando que, no contexto da plataforma VulnFusion, maximizar a deteção de categorias CWE é igualmente importante à consistência dos resultados, e que a diferença deste valor entre o teste 4 e o teste 6 é de apenas alguns pontos percentuais, optou-se por implementar a metodologia desenvolvida no teste 6.

Esta oferece um bom equilíbrio entre cobertura elevada e estabilidade dos resultados, apesar de implicar ligeiro acréscimo nos custos e na complexidade de comunicação com a API.

A melhoria observada na cobertura de mapeamento CWE quando os erros são processados individualmente (como no Teste 3) está relacionada com limitações dos modelos utilizados e com a forma como estes mantêm o contexto de raciocínio. Quando os erros são agrupados em blocos maiores, o modelo revela uma capacidade limitada para manter o contexto, o que pode levar à omissão ou confusão na atribuição de CWEs. Ao enviar cada erro separadamente, evita-se a sobrecarga cognitiva do modelo, permitindo uma análise mais precisa, focada e consistente.

4.2 TESTE DA PLATAFORMA VULNFUSION

O relatório fornecido ao utilizador pela VulnFusion inclui dois atributos, *likelihood* e *risk_score*, que pretendem auxiliar na prioritização das vulnerabilidades a mitigar.

A metodologia de testes descrita nesta secção tem como propósito avaliar a utilidade destes dois indicadores.

4.2.1 *Metodologia*

A Juliet Test Suite - Java disponibiliza um número significativo de testes organizados por categoria CWE, acompanhados de um ficheiro SARIF que contém os resultados esperados e que podem ser utilizados como definição da verdade.

Os testes realizados analisaram casos de estudo fornecidos pela Juliet Test Suite, para CWE específicos, selecionados com base na sua relevância. Devido aos casos de estudo terem sido desenvolvidos em Java, o processo de análise utilizou as ferramentas Semgrep, Spotbugs e PMD.

A análise dos resultados obtidos pela plataforma é realizada em duas etapas distintas. A primeira consiste no mapeamento dos resultados obtidos com os valores esperados pela test suite. Este mapeamento é realizado com base no nome do ficheiro e na linha de código onde o erro foi identificado, aplicando-se uma tolerância de ± 5 linhas, conforme definido previamente no processo de união dos relatórios individuais.

Este processo é implementado através de um script, cujo código relevante se encontra na Listagem 17. O método apresentado percorre todas as ocorrências identificadas (linha 7) e procura realizar a correspondência com os casos da Juliet Test Suite (linhas 12 a 14). Quando é encontrada uma correspondência, o candidato mais próximo é selecionado (linha 20) e a ocorrência é classificada como verdadeiro positivo (TP) ou verdadeiro positivo com erro de classificação (TP_mislabel), dependendo da correspondência da categoria CWE (linha 23). Caso não exista correspondência, a ocorrência é considerada falso positivo (FP) (linha 30). Na linha 38, são identificadas as ocorrências da Juliet Test Suite que não foram mapeadas, sendo classificadas como falsos negativos (FN). Por fim, o método compila todos os resultados num ficheiro CSV (linha 48), que serve de base para a análise estatística e para a avaliação da performance da plataforma VulnFusion.

Listagem 17: Excerto do método que realiza o mapeamento entre os resultados e a verdade

```
def match_dataset_with_line_tolerance(sast_file, juliet_file,
↪  line_tolerance=5):
    # Code related to opening the necessary files
    # Code related to attributes and file name normalization
    matched = []
    matched_juliet_indices = set()

    for idx, row in df_sast.iterrows():
        file = row["file_name"]
        line = row["line"]
        cwe = row.get("cwe")
        # Candidates within line tolerance
        candidates = df_juliet[
            (df_juliet["file_name"] == file) &
            (df_juliet["juliet_line_number"].between(line - line_tolerance,
            ↪  line + line_tolerance))
        ].copy()

        if not candidates.empty:
            # Pick closest match
            candidates["line_diff"] = (candidates["juliet_line_number"] -
            ↪  line).abs()
            best = candidates.sort_values("line_diff").iloc[0]
            matched_juliet_indices.add(best.name)
            # Label as TP or TP_mislabel
            label = "TP" if str(cwe) == str(best["cwe"]) else
            ↪  "TP_mislabel"
            result = row.to_dict()
            for col in df_juliet.columns:
                result[f"juliet_{col}"] = best[col]
            result["label"] = label
        else:
            # No match in Juliet within line range → FP
            result = row.to_dict()
            for col in df_juliet.columns:
                result[f"juliet_{col}"] = None
            result["label"] = "FP"

        matched.append(result)

    # Any Juliet rows not matched → FN
    unmatched_juliet =
    ↪  df_juliet[~df_juliet.index.isin(matched_juliet_indices)]
    for _, row in unmatched_juliet.iterrows():
        result = {col: None for col in df_sast.columns}
        for col in df_juliet.columns:
            result[f"juliet_{col}"] = row[col]
        result["label"] = "FN"
        matched.append(result)
    # Final DataFrame
    merged_df = pd.DataFrame(matched)
    out_name = sast_file.split("-")[0] + "_merged_with_line_tolerance.csv"
    merged_df.to_csv(out_name, index=False)
    print(f"Saved merged results to: {out_name}")

    return merged_df
```

A segunda etapa tem como objetivo extrair métricas que possam ser utilizadas para avaliar o desempenho da VulnFusion em identificar e classificar vulnerabilidades num projeto. Esta avaliação é realizada com base em gamas de valores limite baseados na *likelihood* e *risk score* e descreve três cenários distintos:

- Seleção com base na *likelihood*: filtra os resultados apenas com base na probabilidade de uma vulnerabilidade ser um verdadeiro positivo, para calcular as métricas de classificação e a precisão do mapeamento CWE.
- Seleção com base na *risk score*: filtra os resultados com base na severidade da vulnerabilidade em questão para obter as métricas necessárias.
- Seleção com base em SQI: atributo calculado com base na *likelihood* e no valor normalizado para uma gama de 1-10 do *risk score*. Este novo atributo pode ser representado matematicamente pela formula, abaixo representada, onde os parâmetros α e β podem ser ajustados de forma a otimizar o sistema. Os resultados apresentados nesta secção consideram estes parâmetros com valor 1.

$$\text{SQI} = \alpha L \times \beta R$$

Estes cenários incluem resultados esperados pela Juliet Test Suite que não foram detectados, considerando-os como falsos negativos de forma a assegurar o cálculo correto das métricas de classificação.

A obtenção das métricas de classificação realiza uma correspondência semântica das classes CWE, onde classes relacionadas, como por exemplo CWE-77 e CWE-78 que correspondem a vulnerabilidades relacionadas com injeção de comandos, são agrupadas e consideradas como verdadeiros positivos. Esta abordagem permite que esta análise considere uma variação aceitável na categoria CWE introduzida pelo processo de mapeamento CWE. A tabela 4.9 apresenta os CWEs testados e os respectivos grupos considerados para a análise.

A metodologia de cálculo das métricas de classificação para o parâmetro SQI é implementada no método apresentado na Listagem 18. Nas linhas 5 a 8, realiza-se a normalização dos atributos *likelihood* e *risk_score*, seguida do cálculo do índice composto SQI. A iteração sobre os valores limiar ocorre a partir da linha 14, onde são filtradas as ocorrências com SQI superior ao limiar. A classificação das ocorrências como TP, FP, FN ou TP_mislabel é feita entre as linhas 16 e 22, e o cálculo das métricas de desempenho, *precision*, *recall*, *F1_score* e *CWE accuracy*, é realizado nas linhas 28 a 31.

Tabela 4.9: CWEs testados e correspondência de grupo

CWE	Grupo CWE	Categoria CWE
23	22, 23, 36	*Path traversal*
36	22, 23, 36	*Path traversal*
78	77, 78	Injecção de comandos
89	89, 564	*SQL injection*
134	134	*Format string*
190	190	*Integer Overflow*

Listagem 18: Método de avaliação de métricas para SQI.

```python
def evaluate_sqi_thresholds(merged_file: str, thresholds: list,
    alpha: float = 1.0, beta:  float = 1.0) -> pd.DataFrame:
    df = pd.read_csv(merged_file)
    # Normalize likelihood and risk
    df["L_norm"]    = df["likelihood"] / 100.0
    df["R_norm"]    = 1.0 + df["risk_score"] / 100.0 * 9.0
    # Composite Security Quality Index
    df["SQI"] = alpha * df["L_norm"] * beta * df["R_norm"]
    # Precompute false negatives
    fn_df = df[df["location_file"].isna() &
    ↪  df["juliet_line_number"].notna()].copy()
    fn_df["label"] = "FN"

    results = []
    for thr in thresholds:
        # Predictions above threshold that matched Juliet → tp
        matched = df[(df["SQI"] >= thr) &
            df["juliet_line_number"].notna()].copy()
        matched["label"] = matched.apply(classify_prediction, axis=1)
        # Predictions above threshold with no Juliet match → FP
        fps_df = df[(df["SQI"] >= thr) & df["juliet_line_number"].isna()
        ].copy()
        fps_df["label"] = "FP"
        # Combine and count
        combo = pd.concat([matched, fps_df, fn_df], ignore_index=True)
        cnt = combo["label"].value_counts().to_dict()
        tp, tp_mis = cnt.get("TP", 0), cnt.get("TP_mislabel", 0)
        fp, fn = cnt.get("FP", 0), cnt.get("FN", 0)
        prec     = tp / (tp + fp) if tp + fp else 0.0
        rec      = tp / (tp + fn) if tp + fn else 0.0
        f1_score = 2 * prec * rec / (prec + rec) if prec + rec else 0.0
        cwe_acc  = tp / (tp + tp_mis) if tp + tp_mis else 0.0

        results.append({
            "SQI Threshold": thr, "TP": tp, "TP_mislabel": tp_mis,
            "FP": fp, "FN": fn, "Precision": round(prec, 3),
            "Recall": round(rec, 3), "F1": round(f1_score, 3),
            "CWE Acc": round(cwe_acc, 3)})

    return pd.DataFrame(results)
```

De seguida, apresentam-se os resultados obtidos durante o processo de análise dos CWEs selecionados, com o objetivo de avaliar a eficácia da plataforma VulnFusion na deteção e classificação de vulnerabilidades, esperando-se idealmente valores elevados para as métricas de classificação, em particular na *F1-score* e na *CWE accuracy*. As classificações das ocorrências para os testes realizados podem ser consultadas no Anexo B.

4.2.2 *CWE-23*

A Juliet Test Suite contém 444 casos de estudo referentes ao CWE-23, dos quais se obteve os seguintes resultados.

Tabela 4.10: CWE-23 – Resultados por limiar de probabilidade

Limiar (%)	***Precision***	***Recall***	***F1_score***	***CWE accuracy***
50	7,3%	37,0%	12,2%	82,4%
60	7,4%	37,0%	12,4%	82,4%
70	9,9%	37,0%	15,6%	87,8%
80	22,8%	37,0%	28,2%	93,0%
90	88,9%	1,9%	3,8%	100,0%

Na tabela 4.10, verifica-se que o aumento do limiar de *likelihood* melhora gradualmente a *precision*, atingindo o melhor equilíbrio em 80%, com *F1_score* elevado e *CWE accuracy* de 93%. Com o valor do limiar a 90% a *precision* é alta, contudo a forte queda no *recall* reduz o *F1_score*.

Tabela 4.11: CWE-23 – Resultados por limiar de pontuação de risco

Limiar (%)	***Precision***	***Recall***	***F1_score***	***CWE accuracy***
50	37,8%	37,0%	37,4%	93,7%
60	37,8%	37,0%	37,4%	93,7%
70	37,8%	37,0%	37,4%	93,7%
80	20,2%	9,8%	13,2%	100,0%
90	50,0%	1,5%	2,8%	100,0%

Os valores apresentados na tabela 4.11 demonstram valores consistentes em todas as métricas de classificação para valores de limiar até 70%, inclusive. Com valores

de limiar superiores, verifica-se a alteração no número de ocorrências VP e FP, que resulta numa redução significativa no *F1_score*.

Tabela 4.12: CWE-23 – Resultados por limiar de SQI

Limiar	***Precision***	***Recall***	***F1_score***	***CWE accuracy***
5	37,8%	37,0%	37,4%	93,7%
6	20,3%	10,0%	13,4%	100,0%
7	66,7%	3,3%	6,4%	100,0%
8	0,0%	0,0%	0,0%	0,0%
9	0,0%	0,0%	0,0%	0,0%

A tabela 4.12 apresenta o melhor valor de *F1_score* para o valor de 5. Para valores superiores, observa-se uma redução progressiva do *F1_score*, atingindo valores nulos para limiares superiores a 8. Este comportamento está relacionado com a forma como o SQI combina a probabilidade e o risco, penalizando ocorrências que não apresentam simultaneamente elevada veracidade e impacto.

4.2.3 *CWE-36*

A Juliet Test Suite contém 444 casos de estudo referentes ao CWE-36, dos quais se obteve os seguintes resultados.

Tabela 4.13: CWE-36 – Resultados por limiar de probabilidade

Limiar (%)	***Precision***	***Recall***	***F1_score***	***CWE accuracy***
50	10,6%	37,5%	16,5%	80,7%
60	10,6%	37,5%	16,5%	80,7%
70	15,5%	37,5%	22,0%	89,7%
80	25,7%	37,5%	30,5%	93,8%
90	100,0%	4,3%	8,2%	100,0%

A tabela 4.13 apresenta uma melhoria das métricas *precision* e *F1_score* com o aumento do valor do limiar de *likelihood*, atingindo o melhor ponto de equilíbrio e *CWE accuracy* para o limiar de 80%.

O aumento do valor de limiar de risco mantém valores consistentes até ao limiar de 70%, como demonstrado pela tabela 4.14. O limiar de 80% apresenta uma

Tabela 4.14: CWE-36 – Resultados por limiar de pontuação de risco

Limiar (%)	***Precision***	***Recall***	***F1_score***	***CWE accuracy***
50	38,2%	37,5%	37,9%	93,8%
60	38,2%	37,5%	37,9%	93,8%
70	38,3%	37,5%	37,9%	93,8%
80	36,9%	27,0%	31,2%	96,2%
90	6,9%	0,5%	0,9%	100,0%

redução nas métricas de classificação, incluindo o *F1_score*, mas apresenta um ligeiro aumento na *CWE accuracy*.

Tabela 4.15: CWE-36 – Resultados por limiar de SQI

Limiar	***Precision***	***Recall***	***F1_score***	***CWE accuracy***
5	38,2%	37,5%	37,9%	93,8%
6	34,7%	27,2%	30,5%	90,4%
7	42,6%	4,7%	8,5%	100,0%
8	0,0%	0,0%	0,0%	0,0%
9	0,0%	0,0%	0,0%	0,0%

A utilização do SQI para o CWE-36, apresentada na Tabela 4.15, revela o melhor desempenho global das métricas de classificação para o limiar 5. Para limiares superiores, observa-se uma redução progressiva do *F1_score*, atingindo valores nulos a partir do limiar 8.

4.2.4 *CWE-78*

A Juliet Test Suite contém 444 casos de estudo referentes ao CWE-78, dos quais se obteve os seguintes resultados.

A tabela 4.16 permite observar uma subida do valor de *F1_score*, atingindo o valor máximo de 41.6% para o limiar de *likelihood* de 80%, acompanhado de uma *CWE accuracy* de 93,8%. Para o limiar de 90%, apesar do aumento significativo da *precision*, a redução do *recall* traduz-se numa diminuição do valor de *F1_score*.

Tabela 4.16: CWE-78 – Resultados por limiar de probabilidade

Limiar (%)	***Precision***	***Recall***	***F1_score***	***CWE accuracy***
50	11,2%	41,5%	17,6%	81,0%
60	11,3%	41,5%	17,8%	81,0%
70	13,7%	41,5%	20,5%	86,1%
80	41,7%	41,5%	41,6%	93,8%
90	73,0%	19,9%	31,2%	100,0%

Tabela 4.17: CWE-78 – Resultados por limiar de pontuação de risco

Limiar (%)	***Precision***	***Recall***	***F1_score***	***CWE accuracy***
50	48,5%	41,5%	44,7%	94,7%
60	48,5%	41,5%	44,7%	94,7%
70	49,0%	41,5%	44,9%	95,7%
80	53,8%	41,5%	46,8%	100,0%
90	50,3%	31,7%	38,9%	100,0%

A tabela 4.17 apresenta uma progressão positiva das métricas de classificação, atingindo os melhores valores de *F1_score* e *CWE accuracy* no limiar de risco de 80%. Este resultado indica que, para esta categoria, a severidade da vulnerabilidade é um bom indicador para uma prioritização eficaz.

Tabela 4.18: CWE-78 – Resultados por limiar de SQI

Limiar	***Precision***	***Recall***	***F1_score***	***CWE accuracy***
5	48,5%	41,5%	44,7%	94,7%
6	53,8%	41,5%	46,8%	100,0%
7	56,4%	41,5%	47,8%	100,0%
8	50,0%	0,2%	0,5%	100,0%
9	0,0%	0,0%	0,0%	0,0%

Os resultados obtidos para o SQI, apresentados na tabela 4.18, mostram valores consistentes até ao limiar 7, sendo este o ponto onde se regista o melhor *F1_score*. A partir desse valor, observa-se uma queda acentuada de *F1_score*, atingindo valores nulos.

4.2.5 *CWE-89*

A Juliet Test Suite contém 2220 casos de estudo referentes ao CWE-89, dos quais se obteve os seguintes resultados.

Tabela 4.19: CWE-89 – Resultados por limiar de probabilidade

Limiar (%)	***Precision***	***Recall***	***F1_score***	***CWE accuracy***
50	14,9%	63,6%	24,2%	90,5%
60	15,0%	63,6%	24,3%	90,5%
70	18,1%	63,6%	28,2%	94,4%
80	22,7%	63,6%	33,5%	96,4%
90	26,6%	56,2%	36,1%	100,0%

O limiar de *likelihood* de 90%, apresentado na Tabela 4.19, regista os valores mais elevados de *F1_score* e *CWE accuracy*, apesar de uma ligeira redução na métrica de *recall*. Este resultado indica que, embora o número de ocorrências detetadas seja menor, as que são identificadas apresentam uma elevada veracidade.

Tabela 4.20: CWE-89 – Resultados por limiar de pontuação de risco

Limiar (%)	***Precision***	***Recall***	***F1_score***	***CWE accuracy***
50	23,9%	63,6%	34,7%	97,8%
60	23,9%	63,6%	34,7%	97,8%
70	23,9%	63,6%	34,8%	97,8%
80	24,9%	63,6%	35,8%	100,0%
90	24,6%	47,5%	32,4%	100,0%

A Tabela 4.20 demonstra que os valores de *F1_score* se mantêm relativamente estáveis para os diferentes limiares de risco, com o melhor desempenho registado no limiar de 80%. Este valor coincide com uma *CWE accuracy* de 100%, indicando que as ocorrências identificadas são precisas.

A utilização do índice composto SQI, apresentado na tabela 4.21, revela valores semelhantes para as diversas métricas de classificação utilizadas, atingindo o valor de *F1_score* mais elevado para o limiar de 7. Limiares superiores apresentam uma redução desta métrica, atingindo valores nulos para o limiar de 9.

Tabela 4.21: CWE-89 – Resultados por limiar de SQI

Limiar	***Precision***	***Recall***	***F1_score***	***CWE accuracy***
5	23,9%	63,6%	34,7%	97,8%
6	24,6%	63,6%	35,4%	100,0%
7	25,6%	63,4%	36,5%	100,0%
8	26,2%	31,2%	28,5%	100,0%
9	0,0%	0,0%	0,0%	0,0%

4.2.6 *CWE-134*

A Juliet Test Suite contém 666 casos de estudo referentes ao CWE-134, dos quais se obteve os seguintes resultados.

Tabela 4.22: CWE-134 – Resultados por limiar de probabilidade

Limiar (%)	***Precision***	***Recall***	***F1_score***	***CWE accuracy***
50	7,3%	30,3%	11,8%	74,5%
60	7,3%	30,3%	11,8%	74,5%
70	10,2%	30,3%	15,3%	79,1%
80	15,8%	30,3%	20,7%	82,8%
90	0,0%	0,0%	0,0%	0,0%

A Tabela 4.22 mostra uma melhoria gradual do *F1_score* com o aumento do limiar de *likelihood*, atingindo o melhor valor de 20,7% para o limiar de 80%. Os valores globais das métricas de classificação permanecem baixos em todos os valores de limiar, o que pode indicar limitações na deteção desta categoria por parte das ferramentas utilizadas.

A Tabela 4.23 apresenta valores estáveis para todas as métricas de classificação até ao limiar de 70%. A partir do limiar de 80%, observa-se uma quebra significativa, com todas as métricas a atingir valores nulos, o que indica que a plataforma não considerou as ocorrências desta categoria como vulnerabilidades de elevado risco.

A Tabela 4.24 apresenta resultados positivos apenas para o limiar 5, onde se obtém um *F1_score* de 31,4% e uma *CWE accuracy* de 90,3%. Para limiares superiores, todas as métricas de classificação apresentam valores nulos, o que demonstra que a

Tabela 4.23: CWE-134 – Resultados por limiar de pontuação de risco

Limiar (%)	***Precision***	***Recall***	***F1_score***	***CWE accuracy***
50	32,6%	30,3%	31,4%	90,3%
60	32,6%	30,3%	31,4%	90,3%
70	32,7%	30,3%	31,4%	90,3%
80	0,0%	0,0%	0,0%	0,0%
90	0,0%	0,0%	0,0%	0,0%

Tabela 4.24: CWE-134 – Resultados por limiar de SQI

Limiar	***Precision***	***Recall***	***F1_score***	***CWE accuracy***
5	32,6%	30,3%	31,4%	90,3%
6	0,0%	0,0%	0,0%	0,0%
7	0,0%	0,0%	0,0%	0,0%
8	0,0%	0,0%	0,0%	0,0%
9	0,0%	0,0%	0,0%	0,0%

plataforma não atribuiu simultaneamente elevada veracidade e risco às ocorrências desta categoria, quando avaliadas com o SQI.

4.2.7 *CWE-190*

A Juliet Test Suite contém 666 casos de estudo referentes ao CWE-190. As Tabelas 4.25 a 4.27 demonstram que, para esta categoria, nenhuma das métricas de classificação apresentam resultados positivos, independentemente do limiar aplicado. Estes valores nulos indicam que as ferramentas SAST utilizadas na plataforma VulnFusion não conseguiram identificar ocorrências desta vulnerabilidade.

Tabela 4.25: CWE-190 – Resultados por limiar de probabilidade

Limiar (%)	***Precision***	***Recall***	***F1_score***	***CWE accuracy***
50	0,0%	0,0%	0,0%	0,0%
60	0,0%	0,0%	0,0%	0,0%
70	0,0%	0,0%	0,0%	0,0%
80	0,0%	0,0%	0,0%	0,0%
90	0,0%	0,0%	0,0%	0,0%

Tabela 4.26: CWE-190 – Resultados por limiar de pontuação de risco

Limiar (%)	***Precision***	***Recall***	***F1_score***	***CWE accuracy***
50	0,0%	0,0%	0,0%	0,0%
60	0,0%	0,0%	0,0%	0,0%
70	0,0%	0,0%	0,0%	0,0%
80	0,0%	0,0%	0,0%	0,0%
90	0,0%	0,0%	0,0%	0,0%

Tabela 4.27: CWE-190 – Resultados por limiar de SQI

Limiar	***Precision***	***Recall***	***F1_score***	***CWE accuracy***
5	0,0%	0,0%	0,0%	0,0%
6	0,0%	0,0%	0,0%	0,0%
7	0,0%	0,0%	0,0%	0,0%
8	0,0%	0,0%	0,0%	0,0%
9	0,0%	0,0%	0,0%	0,0%

4.2.8 *Resultados*

Os resultados obtidos a partir da análise dos CWEs selecionados permitem observar que a aplicação de filtros com diferentes valores limite para os parâmetros *likelihood*, *risk_score* e *SQI* apresenta benefícios, que são explorados de seguida com base nas métricas de classificação e precisão obtidas.

A utilização isolada dos parâmetros *likelihood* e *risk_score* apresenta, de modo geral, valores de *F1_score* mais elevados para a gama de valores limite entre 70% e 80%.

O parâmetro SQI, que combina a *likelihood* e *risk_score*, apresenta em média valores de *F1_score* mais elevados na gama de valores limite entre 5-7. Esta métrica composta evidencia um desempenho superior quando comparado com os restantes parâmetros nos CWEs 78 e 89, onde obtém 47.8% e 36.5%, respetivamente. Contudo, para valores limite superiores a 8, estes resultados demonstram uma redução significativa.

Uma análise global dos resultados permite concluir que não existe um valor limite universal que otimize os resultados obtidos para todos os parâmetros em simultâneo. Contudo, a aplicação de filtros com base no parâmetro SQI e com um valor limite aproximadamente 6, permite obter valores que oferecem consistentemente um bom ponto de equilíbrio entre a *precision* e *recall*. A introdução das variáveis configuráveis no parâmetro SQI poderá contribuir para uma otimização destes resultados.

Os resultados obtidos para o CWE-190 apresentam um caso excecional, onde se obteve o valor zero para todos os parâmetros e para todos os valores limite testados. Este resultado evidencia uma limitação nas ferramentas SAST utilizadas (Semgrep CE, SpotBugs e PMD) na deteção de vulnerabilidades relacionadas com *integer overflow*.

A análise dos resultados por categoria CWE permite observar que as ferramentas SAST integradas na VulnFusion tiveram melhor desempenho nas vulnerabilidades de *path traversal*, *command injection*, *SQL injection* e *format string*. Este desempenho está relacionado com o facto de estas vulnerabilidades estarem bem representadas nas regras das ferramentas utilizadas. Por outro lado, no caso do *integer overflow*, não foi detetada qualquer ocorrência, o que se explica pela natureza desta vulnerabilidade, que depende fortemente de valores em tempo de execução e não está devidamente coberta pelas regras das ferramentas *open-source* selecionadas.

Quanto ao enriquecimento por IA, verificou-se que este trouxe ganhos significativos, sobretudo nas vulnerabilidades de *command injection* e *SQL injection*, cujas descrições e contextos são ricos em informação semântica que os modelos conseguem interpretar. Em contraste, vulnerabilidades como *format string*, que apresentam sinais mais subtis e dependem de detalhes específicos do código, beneficiaram menos do processo de enriquecimento.

No contexto da plataforma VulnFusion, a limitação demonstrada pelo CWE-190 indica a necessidade de integrar ferramentas adicionais capazes de identificar

este tipo de vulnerabilidades, ou de adicionar regras especializadas às ferramentas atualmente em uso.

A análise comparativa, apresentada na tabela 4.28, evidencia que a plataforma VulnFusion desenvolvida neste projeto apresenta resultados competitivos face ao estado da arte. Enquanto em [1] os autores demonstraram uma cobertura inferior a 50% no Juliet Test Suite, com uma única ferramenta a identificar apenas 11% dos CWEs, a presente plataforma, ao combinar quatro ferramentas *open-source* e aplicar enriquecimento por IA, alcançou praticamente 100% de cobertura nos testes sobre o VulnerableApp (Teste 6), ainda que com variações em cenários de maior complexidade. Resultados semelhantes foram observados em [3], que comprovaram os ganhos da combinação multi-ferramenta, mas a integração proposta nesta tese vai mais além ao reduzir falsos positivos através de métricas de probabilidade e risco, nomeadamente com o indicador SQI, que obteve F1-scores superiores (47,8% no CWE-78 e 36,5% no CWE-89). Tal aproxima-se da abordagem observada em [6], onde os autores utilizaram AdaBoost para hierarquizar alertas com uma precisão de 0,8, mas distingue-se pela utilização de modelos de linguagem generalistas para enriquecer os resultados. Por outro lado, a limitação observada no CWE-190 (0% de deteção) confirma a conclusão de Nunes em [4], de que nem sempre a combinação de ferramentas assegura cobertura total, reforçando a necessidade de estratégias complementares de deteção.

Em síntese, os resultados demonstram que a integração multi-ferramenta aliada ao enriquecimento por IA contribui para melhorar a cobertura e reduzir falsos positivos, posicionando a VulnFusion como um avanço relevante em relação aos *benchmarks* analisados. A tabela 4.28 consolida os resultados dos estudos de referência e os resultados obtidos pela plataforma apresentada.

4.3 SÍNTESE

Este capítulo apresenta os testes realizados para validar a eficácia da plataforma VulnFusion. São descritas as metodologias utilizadas para avaliar o processo de mapeamento de categorias CWE e a classificação das ocorrências com base em métricas de probabilidade e risco com recurso à API da OpenAI. Os resultados demonstram que a integração de múltiplas ferramentas SAST, aliada ao enriquecimento por inteligência artificial, contribui para uma melhoria na cobertura e precisão da deteção de vulnerabilidades, especialmente em categorias como *command injection* e *SQL*

Tabela 4.28: Resumo de estudos e ferramentas de análise de vulnerabilidades

Estudo / Ferramenta	**Conjunto de Teste**	**Nº de Ferramentas**	**Taxa de Deteção / Cobertura**	**Falsos Positivos**	**Observações**
[1]	Juliet Test Suite (Java, 112 CWEs)	6 open-source + 2 comerciais	< 50% global; melhor ferramenta: 11% CWEs	Baixos	Cada ferramenta melhor num CWE específico.
[3]	SAP dataset (NVD, código real vulnerável)	4 ferramentas	11,2% – 26,5% individual; 38,3% combinado	Não detalhado	Configuração customizada do Semgrep aumentou cobertura de 15,3% → 44,7%.
[4]	134 plugins WordPress	5 gratuitas	Até >70% TPR em cenário de qualidade máxima	Varia; trade-off FP/FN	Combinação nem sempre melhor.
[6]	Juliet Test Suite (v1.2)	3 open-source	Precisão do modelo: 0,8	Redução significativa	AdaBoost p/ classificar e hierarquizar alertas.
VulnFusion (tese, 2025)	Juliet Test Suite (CWEs 23,36,78,89,134,190) + VulnerableApp (62 erros)	4 open-source + IA	≈100% cobertura CWE (Teste 6, VulnerableApp); Juliet: bons resultados exceto CWE-190 (0%)	Redução via IA (melhor F1: 47,8% CWE-78; 36,5% CWE-89 com SQI)	Integração multi-ferramenta + enriquecimento IA; GUI modular e escalável.

injection. Foram também identificadas limitações em determinadas categorias, como *integer overflow*.

O capítulo seguinte apresenta as conclusões retiradas após a realização do trabalho descrito neste documento, incluindo uma análise crítica sobre os objetivos definidos no Capítulo 1. Adicionalmente, descreve pontos de melhoria e evolução para a plataforma VulnFusion.

5

CONCLUSÕES

O presente projeto partiu da constatação de que, apesar da relevância crescente das ferramentas SAST no desenvolvimento seguro de software, a sua adoção continua limitada por dois fatores principais: a reduzida cobertura individual de vulnerabilidades e a elevada taxa de falsos positivos. Para colmatar estas lacunas, foi desenvolvida a plataforma VulnFusion, modular e extensível, que integra quatro ferramentas *open-source* (Semgrep, SpotBugs/FindSecBugs, PMD e Cppcheck), uniformiza os resultados em SARIF, aplica algoritmos de fusão para eliminar redundâncias e recorre a inteligência artificial para enriquecer os achados com métricas adicionais de risco e probabilidade.

Os resultados obtidos confirmam avanços relevantes. Nos testes com o VulnerableApp, foi alcançada cobertura próxima dos 100% em cenários de maior granularidade, enquanto nos testes com o Juliet Test Suite a VulnFusion demonstrou bons níveis de precisão e recall em CWEs críticos como *path traversal*, *command injection* e *SQL injection*. A introdução do indicador SQI permitiu melhorar a priorização de vulnerabilidades e reduzir falsos positivos, em linha com tendências recentes da literatura.

Contudo, persistem limitações importantes. O CWE-190 (*integer overflow*) não foi detetado em nenhum dos casos, revelando a dificuldade das ferramentas selecionadas em lidar com vulnerabilidades dependentes de valores em tempo de execução. Esta limitação está documentada também em trabalhos anteriores, como [1], o que reforça que o problema é estrutural e não específico da plataforma VulnFusion. Além disso, a avaliação baseou-se em conjuntos de dados controlados (Juliet e VulnerableApp), faltando ainda validação em projetos de maior dimensão e em diferentes linguagens, o que condiciona a generalização dos resultados.

Outro aspeto a considerar é a dependência de um serviço externo de inteligência artificial. O recurso à API da OpenAI trouxe ganhos claros na classificação e priorização, mas levanta questões de custo, privacidade e reprodutibilidade, que deverão ser exploradas em trabalhos futuros.

Em síntese, este projeto contribui para o avanço do estado da arte ao demonstrar que a integração de múltiplas ferramentas *open-source*, combinada com o enrique-

cimento por IA, pode melhorar a cobertura de vulnerabilidades e reduzir falsos positivos. Embora não resolva todos os problemas associados à análise estática, o trabalho abre caminho para futuras investigações e aplicações em contextos DevSecOps, onde a escalabilidade, a diversidade de linguagens e a confiança nos resultados permanecem desafios centrais.

5.1 ANÁLISE AOS OBJETIVOS DE INVESTIGAÇÃO

O trabalho de investigação descrito neste documento teve como base quatro objetivos principais, apresentados previamente no Capítulo 1. De seguida, apresenta-se uma análise crítica sobre o cumprimento dos mesmos.

O primeiro objetivo, desenvolver uma plataforma de integração de ferramentas SAST *open-source*, foi cumprido através da criação da plataforma VulnFusion, capaz de integrar múltiplas ferramentas de código aberto. A arquitetura modular e extensível demonstrou ser adequada para a execução coordenada das ferramentas selecionadas. A VulnFusion não apresenta custos de licenciamento das ferramentas SAST, contudo, apresenta custos associados à utilização da API da OpenAI. O processo de adição de novas ferramentas ficou aquém do esperado, sendo atualmente necessárias ligeiras alterações de código para adicionar novas ferramentas.

O segundo objetivo, aumentar a precisão na deteção de vulnerabilidades, também foi alcançado. Os testes realizados confirmaram que a combinação de diferentes ferramentas SAST permitiu uma melhoria da cobertura e da precisão na deteção de vulnerabilidades, em comparação com a execução isolada de cada ferramenta. Ainda assim, algumas categorias de vulnerabilidades apresentaram resultados inferiores ao esperado, o que evidencia que a fusão das ferramentas selecionadas não elimina totalmente as limitações individuais de cada uma.

O terceiro objetivo, implementar uma metodologia de agregação de resultados, foi concretizado através da utilização do formato SARIF e da implementação de um processo de fusão de relatórios. Este processo permitiu consolidar os resultados num único relatório, estruturado e de fácil interpretação para o utilizador. Contudo, verificou-se que os relatórios em formato SARIF, por vezes, não apresentam a mesma informação quando comparados com outros formatos, e que alguns casos específicos de sobreposição de resultados poderiam beneficiar de otimização adicional nos critérios de fusão.

O quarto objetivo, integrar inteligência artificial, foi igualmente atingido. A utilização da API da OpenAI permitiu enriquecer os relatórios com classificações CWE, métricas de *likelihood* e *risk-score*, bem como recomendações de mitigação. Apesar dos resultados positivos, a dependência de um serviço externo e os custos associados podem constituir limitações à adoção da solução apresentada.

De forma geral, os objetivos definidos no Capítulo 1 foram alcançados. A plataforma VulnFusion demonstrou melhorias na deteção e priorização de vulnerabilidades. No entanto, alguns aspetos, como a dependência de serviços externos de inteligência artificial e a necessidade de ajustes no processo de fusão de relatórios, apresentam oportunidades de melhoria.

A secção seguinte descreve estas e outras oportunidades de melhoria identificadas a realizar em trabalhos futuros.

5.2 TRABALHO FUTURO

O presente documento descreve o desenvolvimento e as funcionalidades de um protótipo que permite a análise de projetos com múltiplas ferramentas SAST em simultâneo, e enriquecer os resultados com recurso a técnicas de inteligência artificial. Sendo um protótipo, existem amplas oportunidades de evolução, tanto ao nível de melhorias de funcionalidades existentes como a adição de novas funcionalidades. A presente secção aborda algumas das melhorias identificadas para trabalhos futuros.

Interface e Experiência do Utilizador

Uma das melhorias identificadas incide sobre o *frontend*, com o objetivo de tornar o design mais apelativo e melhorar a forma como os resultados da análise são apresentados ao utilizador. Adicionalmente, a inclusão de informação relevante, como o nome do projeto analisado e das ferramentas selecionadas, permite contribuir para uma experiência mais informativa e personalizada.

Análise com ferramentas SAST

O processo de análise com recurso a múltiplas ferramentas SAST é realizado de forma sequencial, o que pode aumentar significativamente o tempo de execução. A implementação de paralelismo no processo de análise iria permitir reduzir este tempo, contudo alguns mecanismos de controlo devem ser considerados para uma correta gestão dos recursos de sistema disponíveis tendo em conta o número de ferramentas

SAST em questão. Atualmente, os comandos utilizados para a execução de cada ferramenta encontram-se definidos no código, o que retira ao utilizador a opção de parametrizar a utilização de cada ferramenta. Permitir a parametrização destes comandos dá ao utilizador um maior controlo sobre as ferramentas da VulnFusion.

União de relatórios

A etapa de "União de relatórios"apresenta também diversas oportunidades de melhoria. O algoritmo de iteração das ocorrências pode ser melhorado, para reduzir o tempo de execução e o consumo de memória. Além disso, os critérios de igualdade entre ocorrências encontram-se fixos no código, sem possibilidade de ajuste por parte do utilizador. A configuração destes valores permitiria adaptar o grau de tolerância da comparação conforme a necessidade do utilizador. Sugere-se ainda a utilização de técnicas de inteligência artificial para analisar o contexto dos identificadores de erro, permitindo identificar semelhanças semânticas que não são captadas por comparações baseadas apenas em *tokens*.

Mapeamento CWE

Uma possível melhoria ao processo de mapeamento CWE consiste na implementação de uma metodologia adaptativa em três fases, com o objetivo de aumentar a precisão, reduzir custos financeiros e melhorar a interpretabilidade dos resultados. Esta nova abordagem consiste em três fases:

- Classificação em *batch* das ocorrências: as ocorrências são agrupados e enviados ao modelo, que devolve, para cada ocorrência, a categoria CWE e um valor de confiança associado à correspondência.
- Verificação adaptativa: com base num limiar de confiança definido, as ocorrências com valores inferiores são submetidas a uma verificação mais rigorosa.
- Validação semântica: avalia a equivalência entre categorias CWE ou a similaridade semântica entre descrições, em vez de depender exclusivamente da correspondência exata de identificadores.

BIBLIOGRAFIA

[1] M. Esposito, V. Falaschi e D. Falessi, «An Extensive Comparison of Static Application Security Testing Tools», en, 2024. DOI: 10.13140/RG.2.2.12326.54085. URL: https://rgdoi.net/10.13140/RG.2.2.12326.54085.

[2] K. Kuszczyński e M. Walkowski, «Comparative Analysis of Open-Source Tools for Conducting Static Code Analysis», *Sensors*, vol. 23, n.º 18, p. 7978, set. de 2023, ISSN: 1424-8220. DOI: 10.3390/s23187978. URL: http://dx.doi.org/10.3390/s23187978.

[3] G. Bennett, T. Hall, E. Winter e S. Counsell, «Semgrep*: Improving the Limited Performance of Static Application Security Testing (SAST) Tools», em *Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering*, sér. EASE 2024, ACM, jun. de 2024, pp. 614–623. DOI: 10.1145/3661167.3661262. URL: http://dx.doi.org/10.1145/3661167.3661262.

[4] P. Nunes, I. Medeiros, J. Fonseca, N. Neves, M. Correia e M. Vieira, «On Combining Diverse Static Analysis Tools for Web Security: An Empirical Study», em *2017 13th European Dependable Computing Conference (EDCC)*, IEEE, set. de 2017, pp. 121–128. DOI: 10.1109/edcc.2017.16. URL: http://dx.doi.org/10.1109/EDCC.2017.16.

[5] L. Flynn et al., «Prioritizing alerts from multiple static analysis tools, using classification models», em *Proceedings of the 1st International Workshop on Software Qualities and Their Dependencies*, sér. ICSE '18, ACM, mai. de 2018, pp. 13–20. DOI: 10.1145/3194095.3194100. URL: http://dx.doi.org/10.1145/3194095.3194100.

[6] A. Ribeiro, P. Meirelles, N. Lago e F. Kon, «Ranking warnings from multiple source code static analyzers via ensemble learning», em *Proceedings of the 15th International Symposium on Open Collaboration*, sér. OpenSym '19, ACM, ago. de 2019, pp. 1–10. DOI: 10.1145/3306446.3340828. URL: http://dx.doi.org/10.1145/3306446.3340828.

[7] G. Bennett, T. Hall, S. Counsell, E. Winter e T. Shippey, «Do Developers Use Static Application Security Testing (SAST) Tools Straight Out of the Box? A large-scale Empirical Study», em *Proceedings of the 18th ACM/IEEE Inter-*

national Symposium on Empirical Software Engineering and Measurement, sér. ESEM '24, ACM, out. de 2024, pp. 454–460. DOI: 10.1145/3674805.3690750. URL: http://dx.doi.org/10.1145/3674805.3690750.

[8] OASIS, *Static Analysis Results Interchange Format (SARIF) Version 2.1.0 — oasis-open.org*, https://www.oasis-open.org/standard/sarifv2-1-os/?utm_source=chatgpt.com, [Accessed 17-09-2025], 2025.

[9] S. Kummita e G. Piskachev, «Integration of the Static Analysis Results Interchange Format in CogniCrypt», jul. de 2019. arXiv: 1907.02558 [cs.PL].

[10] Policygenius e J. Grider, *Using SARIF to Automate Vulnerability Remediation Tracking — linkedin.com*, https://www.linkedin.com/pulse/using-sarif-automate-vulnerability-remediation-tracking-, [Accessed 18-09-2025], 2022.

[11] GitHub, *github merge-results - GitHub Docs — docs.github.com*, https://docs.github.com/en/code-security/codeql-cli/codeql-cli-manual/github-merge-results?utm_source=chatgpt.com, [Accessed 18-09-2025], 2025.

[12] Semgrep, *GitHub - semgrep/semgrep: Lightweight static analysis for many languages. Find bug variants with patterns that look like source code. — github.com*, https://github.com/semgrep/semgrep, [Accessed 02-07-2025], 2025.

[13] Semgrep, *Semgrep AppSec Platform versus Community Edition | Semgrep — semgrep.dev*, https://semgrep.dev/docs/semgrep-pro-vs-oss, [Accessed 02-07-2025], 2025.

[14] Semgrep, *CLI reference | Semgrep — semgrep.dev*, https://semgrep.dev/docs/cli-reference, [Accessed 02-07-2025], 2025.

[15] Spotbugs, *Introduction; spotbugs 4.9.3 documentation — spotbugs.readthedocs.io*, https://spotbugs.readthedocs.io/en/stable/introduction.html, [Accessed 04-07-2025], 2025.

[16] Spotbugs, *Bug descriptions; spotbugs 4.9.3 documentation — spotbugs.readthedocs.io*, https://spotbugs.readthedocs.io/en/stable/bugDescriptions.html, [Accessed 04-07-2025], 2025.

[17] FindSecBugs, *Find Security Bugs — find-sec-bugs.github.io*, https://find-sec-bugs.github.io/, [Accessed 04-07-2025], 2025.

[18] PMD, *Documentation Index | PMD Source Code Analyzer — docs.pmd-code.org*, https://docs.pmd-code.org/latest/, [Accessed 05-07-2025], 2025.

[19] PMD, *PMD CLI reference | PMD Source Code Analyzer — docs.pmd-code.org*, https://docs.pmd-code.org/latest/pmd_userdocs_cli_reference.html, [Accessed 05-07-2025], 2025.

[20] Cppcheck, *Cppcheck - A tool for static C/C++ code analysis — cppcheck.sourceforge.io*, https://cppcheck.sourceforge.io/, [Accessed 05-07-2025], 2025.

[21] OASIS, *Static Analysis Results Interchange Format (SARIF) Version 2.1.0 Plus Errata 01 — docs.oasis-open.org*, https://docs.oasis-open.org/sarif/sarif/v2.1.0/errata01/os/sarif-v2.1.0-errata01-os-complete.html, [Accessed 18-08-2025], 2025.

[22] M. Fowler, *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional, 2003, ISBN: 9780321127426. URL: https://books.google.com/books/about/Patterns_of_Enterprise_Application_Archi.html?hl=&id=Jl5rkQnbfAIC.

[23] RapidFuzz, *GitHub - rapidfuzz/RapidFuzz: Rapid fuzzy string matching in Python using various string metrics — github.com*, https://github.com/rapidfuzz/RapidFuzz, [Accessed 10-08-2025], 2025.

[24] SasanLabs, *GitHub - SasanLabs/VulnerableApp: OWASP VulnerableApp Project: For Security Enthusiasts by Security Enthusiasts. — github.com*, https://github.com/SasanLabs/VulnerableApp, [Accessed 20-07-2025], 2025.

[25] OpenAi, *OpenAI Platform — platform.openai.com*, https://platform.openai.com/docs/faq/how-should-i-set-the-temperature-parameter, [Accessed 28-09-2025], 2025.

APÊNDICE A - *PROMPTS* DOS TESTES MAPEAMENTO CWE

A.1 TESTE 1

Listagem 19: Mapeamento CWE - Teste 1 *prompt* 1

```
Here is a list of SAST tool errors. For each error, identify the corresponding CWE, provide a short description of the CWE, and suggest one or two possible solutions:
{', '.join(chunk)}

Response Requirements:
- **Only include errors that have a corresponding CWE.** If no match is found, omit the error.
- **Format the response as a valid JSON array** containing objects with the following structure:
  [
    {{
      "sast_error": "<SAST_ERROR>",
      "cwe": "<CWE_IDENTIFIER>",
      "cwe_description": "<CWE_DESCRIPTION>",
      "solutions": [
        "<SOLUTION_1>",
        "<SOLUTION_2>"
      ]
    }},
    ...
  ]
- **No additional text, labels, markdown syntax (` ``` `), or extra characters.**
- **Ensure responses do NOT include the semicolon (`;`) character.**
- **Avoid trailing commas in JSON to ensure validity.**
- Solutions should be based on industry best practices (where applicable).
```

Listagem 20: Mapeamento CWE - Teste 1 *prompt* 2

```
You are an expert in application security and vulnerability analysis, specializing in CWE classifications based on OWASP and MITRE standards.

Below is a list of SAST tool errors. For each error, identify the **single most accurate CWE**, provide a **consistent** short description of the CWE, and suggest one or two **industry-standard solutions**.

### Error List:
{', '.join(chunk)}

### Response Requirements:
- **Always return the SAME CWE for the SAME error.** If an error has multiple valid CWEs, select the one most widely accepted by industry standards.
- **Reference MITRE CWE classifications strictly to ensure accuracy.** If no CWE exists for an error, **omit it from the response**.
- **Format the response as a valid JSON array**, following this structure:
  [
    {{
      "sast_error": "<SAST_ERROR>",
      "cwe": "<CWE_IDENTIFIER>",
      "cwe_description": "<CWE_DESCRIPTION>",
      "solutions": [
        "<SOLUTION_1>",
        "<SOLUTION_2>"
      ]
    }},
  ]

- **Follow OWASP & MITRE CWE mappings precisely; avoid subjective CWE assignments.**
- **No additional text, markdown (```), or extra characters in the response.**
- **Ensure responses do NOT include the semicolon (;) character.**
- **Avoid trailing commas in JSON to ensure validity.**
- **Use the same sentence structure for CWE descriptions across responses.**
- **Format solutions concisely, using standardized phrasing and terminology.**
```

Listagem 21: Mapeamento CWE - Teste 1 *prompt* 3

```
You are an expert in application security and vulnerability analysis, specializing in CWE classifications based on OWASP and MITRE standards.

You will receive a list of static analysis tool (SAST) error identifiers. For each error, return the **single most accurate CWE** classification, a **standardized CWE description**, and **one or two concise, industry-standard solutions**.

### Requirements:
- Use **MITRE CWE** as the authoritative source.
- If an error maps to multiple CWEs, choose the **most widely accepted** one.
- If no valid CWE exists, **omit the error** from the response.
- **Always return the same CWE for the same error**, regardless of context.
- Do not infer or guess; only respond if the mapping is clear and standard.
- Ensure **no semicolons**, **no markdown**, and **no trailing commas**.
- Format the output as a **valid JSON array** using this structure:

[
  {{
    "sast_error": "<SAST_ERROR>",
    "cwe": "<CWE_IDENTIFIER>",
    "cwe_description": "<Standard CWE description>",
    "solutions": [
      "<Solution 1>",
      "<Solution 2>"
    ]
  }},
]

### Error List:
{', '.join(chunk)}
```

A.2 TESTE 2 E 3

Listagem 22: Mapeamento CWE - Teste 2 e 3 *prompt* 1

```
You are a security expert with deep knowledge of static analysis tools (SAST), CWE mappings, and secure coding best practices.

Below is a list of SAST errors. Each error is represented by an identifier and a tool-generated message in the format:
<ERROR_ID>: <DESCRIPTION>

Your task:
For each error, identify the **single most accurate CWE classification**, provide a **concise and standardized CWE description**, and suggest **one or two clear, industry-standard solutions**.

### Errors:
{chr(10).join([f"{k}: {v}" for k, v in chunk.items()])}

### Response Format:
- **Only include errors that have a valid CWE mapping.** Omit any without a clear match.
- **Return a valid JSON array** using this exact structure:
  [
    {{
      "sast_error": "<ERROR_ID>",
      "cwe": "<CWE_IDENTIFIER>",
      "cwe_description": "<CWE_DESCRIPTION>",
      "solutions": [
        "<SOLUTION_1>",
        "<SOLUTION_2>"
      ]
    }},
  ]

### Important Constraints:
- Descriptions are taken directly from the SAST tool; interpret them accordingly.
- Do not invent or guess CWE mappings. Be strict in following MITRE's definitions.
- Use consistent sentence structure for CWE descriptions.
- Use concise, actionable, best-practice-based phrasing for solutions.
- **Do NOT include any extra text**, such as comments, headers, or markdown.
- **No semicolons** and **no trailing commas** in the JSON.
```

Listagem 23: Mapeamento CWE - Teste 2 e 3 *prompt* 2

```
You are an expert in application security and vulnerability analysis, with deep knowledge of CWE classifications following OWASP and MITRE standards.

Below is a list of SAST tool errors. Each error includes an identifier and a tool-generated description.

For each error:
- Identify the **single most accurate CWE** based on MITRE's official taxonomy.
- Provide a **clear, standardized CWE description**.
- Recommend **one or two concise, industry-standard solutions**.

### Error List:
{chr(10).join([f"{k}: {v}" for k, v in chunk.items()])}

### Response Format:
Return a **valid JSON array**, with each object using this structure:
[
  {{
    "sast_error": "<ERROR_ID>",
    "cwe": "<CWE_IDENTIFIER>",
    "cwe_description": "<CWE_DESCRIPTION>",
    "solutions": [
      "<SOLUTION_1>",
      "<SOLUTION_2>"
    ]
  }},
]

### Output Requirements:
- **Only include errors with a valid CWE match.** Omit unmatched errors.
- **Use the exact same CWE for the same error every time.** Prioritize widely accepted mappings.
- **Do not guess or infer CWE mappings.** Follow MITRE strictly.
- **Avoid subjective or loosely related CWEs.**
- Ensure the output is **valid JSON**:
  - No additional text or formatting (e.g., markdown).
  - No semicolons (`;`).
  - No trailing commas.

- Keep descriptions and solutions **uniform, professional, and phrased with industry terminology**.
```

Listagem 24: Mapeamento CWE - Teste 2 e 3 *prompt* 3

```
You are an expert in application security and vulnerability analysis, specializing in CWE classifications based on OWASP and MITRE standards.

You will be given a list of static analysis tool (SAST) errors. Each error includes an identifier and a tool-generated description.

For each error:
- Identify the **single most accurate CWE** classification (based on MITRE CWE).
- Provide a **standardized, concise CWE description**.
- Suggest **one or two brief, industry-standard solutions**.

### Error List:
{chr(10).join([f"{k}: {v}" for k, v in chunk.items()])}

### Response Format:
Return a **valid JSON array** using the structure below:
[
  {{
    "sast_error": "<ERROR_ID>",
    "cwe": "<CWE_IDENTIFIER>",
    "cwe_description": "<CWE_DESCRIPTION>",
    "solutions": [
      "<SOLUTION_1>",
      "<SOLUTION_2>"
    ]
  }},
  ...
]

### Response Rules:
- **Use MITRE CWE as the sole source of truth.**
- If an error maps to multiple CWEs, choose the **most widely accepted** one.
- **Omit errors with no clear or official CWE mapping.**
- **Always return the same CWE for the same error.** No contextual variation.
- **Do not infer or guess.** Only respond where the mapping is well-established.
- Output must be **clean, valid JSON**:
  - No extra text or formatting (e.g., markdown).
  - No semicolons (`;`) anywhere.
  - No trailing commas.

- Keep language **professional, consistent, and aligned with security best practices**.
```

A.3 TESTE 4

Listagem 25: Mapeamento CWE - Teste 4 *prompt* primeiro pedido instrução *system*

```
You are an expert in application security and vulnerability analysis with deep knowledge
of MITRE CWE classifications.
For each SAST error, identify the most accurate CWE identifier and its official description,
and recommend 1-2 concise, industry-standard solutions.
### RESPONSE RULES:
- Return **only** a valid JSON array of objects.
- Use **double quotes** for all keys and string values.
- No markdown, headings, or extraneous text.
- No semicolons (`;`). No trailing commas.
- Skip any error if you cannot confidently map it.
- Maintain consistent mappings for duplicate errors.
```

Listagem 26: Mapeamento CWE - Teste 4 *prompt* primeiro pedido instrução *user*

```
### Error List:
+ "\n".join(f"{k}: {v}" for k, v in chunk.items())
+ "\n\n### Output format:
  Return exactly a JSON array where each object has these keys, in this order:
    sast_error, cwe, cwe_description, solutions
  - `sast_error`: the error ID string
  - `cwe`: official CWE identifier (e.g., \"CWE-79\")
  - `cwe_description`: official CWE title
  - `solutions`: array of 1-2 solution strings
  No extra keys or text."
```

Listagem 27: Mapeamento CWE - Teste 4 *prompt* segundo pedido instrução *system*

```
You are a security analyst. You will receive a JSON array of SAST findings
with initial CWE assignments. Your job is to validate or correct each
↪ mapping.
For each item:
  • If the CWE is correct:
    - set \"verified\": true
    - keep the existing \"cwe\", \"cwe_description\", and \"solutions\"
  • If the CWE is incorrect:
    - set \"verified\": false
    - replace \"cwe\" with the correct identifier (e.g. \"CWE-22\")
    - provide the official \"cwe_description\
    - add \"justification\" (1-2 sentences)\
    - suggest 1-2 updated \"solutions\"
Response rules:
  - Return only a valid JSON array of the same length and order.
  - Each object's keys must appear in this exact order:
      sast_error, verified, cwe, cwe_description, justification, solutions
  - Use double quotes only; no single quotes, semicolons, trailing commas,
    markdown, comments, or extra fields.
  - Do not repeat justification text across entries.
  - If you cannot determine correctness, output:
    {\"sast_error\":\"<ERROR_ID>\",\"verified\":false,
\"cwe\":\"\",\"cwe_description\":\"\",
\"justification\":\"CWE mapping unclear. Needs manual review.\",
\"solutions\":[]}
```

A.4 TESTE 5

Listagem 28: Mapeamento CWE - Teste 5 *prompt* primeiro pedido instrução *system*

```
You are an expert in application security and vulnerability analysis with
↪ deep knowledge
of MITRE CWE classifications.
For each SAST error, identify the most accurate CWE identifier and its
↪ official description,
and recommend 1-2 concise, industry-standard solutions.
### RESPONSE RULES:
- Return **only** a valid JSON array of objects.
- Use **double quotes** for all keys and string values.
- No markdown, headings, or extraneous text.
- No semicolons (`;`). No trailing commas.
- Skip any error if you cannot confidently map it.
- Maintain consistent mappings for duplicate errors.
```

Listagem 29: Mapeamento CWE - Teste 5 *prompt* primeiro pedido instrução *user*

```
### Error List:
+ "\n".join(f"{k}: {v}" for k, v in chunk.items())
+ "\n\n### Output format:
  Return exactly a JSON array where each object has these keys, in this
  ↪ order:
    sast_error, cwe, cwe_description, solutions
  - `sast_error`: the error ID string
  - `cwe`: official CWE identifier (e.g., \"CWE-79\")
  - `cwe_description`: official CWE title
  - `solutions`: array of 1-2 solution strings
  No extra keys or text."
```

Listagem 30: Mapeamento CWE - Teste 5 *prompt* segundo pedido instrução *system*

```
You are a security analyst. You will receive a JSON array of SAST findings
with initial CWE assignments. Your job is to validate or correct each
↪ mapping.
For each item:
  • If the CWE is correct:
    - set \"verified\": true
    - keep the existing \"cwe\", \"cwe_description\", and \"solutions\"
  • If the CWE is incorrect:
    - set \"verified\": false
    - replace \"cwe\" with the correct identifier (e.g. \"CWE-22\")
    - provide the official \"cwe_description\
    - add \"justification\" (1-2 sentences)\
    - suggest 1-2 updated \"solutions\"
Response rules:
  - Return only a valid JSON array of the same length and order.
  - Each object's keys must appear in this exact order:
      sast_error, verified, cwe, cwe_description, justification, solutions
  - Use double quotes only; no single quotes, semicolons, trailing commas,
    markdown, comments, or extra fields.
  - Do not repeat justification text across entries.
  - If you cannot determine correctness, output:
    {\"sast_error\":\"<ERROR_ID>\",\"verified\":false,
\"cwe\":\"\",\"cwe_description\":\"\",
\"justification\":\"CWE mapping unclear. Needs manual review.\",
\"solutions\":[]}
```

Listagem 31: Mapeamento CWE - Teste 5 *prompt* terceiro pedido instrução *system*

```
You are a senior security architect with deep expertise in MITRE CWE.
Your task is to triple-check each SAST finding's CWE mapping, based solely
on the initial AI mapping and the first verification pass.
```

Listagem 32: Mapeamento CWE - Teste 5 *prompt* terceiro pedido instrução *user*

```
You will receive a JSON array of objects, each containing two sub-objects:
  • \"initial\": the original AI mapping with keys:
      - sast_error
      - sast_message
      - cwe
      - cwe_description
      - solutions
  • \"verified\": the first verification pass with keys:
      - verified (true/false)
      - cwe
      - cwe_description
      - justification
      - solutions
Your output **must** be a JSON array of objects, each with these keys **in this exact order**:
1. \"sast_error\"
2. \"cwe\"
3. \"cwe_description\"
4. \"solutions\"
5. \"verified_third_pass\" (true/false)
**Instructions:**
- For each finding, independently reassess the CWE. You may reuse the same CWE or pick a different one if warranted.
- Provide the official CWE description.
- Recommend 1-2 concise, industry-standard solutions.
- Set \"verified_third_pass\" to true only if you are highly confident this CWE is correct; otherwise false.
- If you cannot be certain, set verified_third_pass=false and leave cwe and cwe_description empty.
**Formatting rules:** only return valid JSON. Use double quotes, no comments, no markdown, no semicolons, no extra fields, no trailing commas.
f"Input data:\n{json.dumps(payload, indent=2)}
```

A.5 TESTE 6

Listagem 33: Mapeamento CWE - Teste 6 *prompt* primeiro pedido instrução *system*

```
You are an expert in application security and vulnerability analysis with deep knowledge
of MITRE CWE classifications.
For each SAST error, identify the most accurate CWE identifier and its official description,
and recommend 1-2 concise, industry-standard solutions.
### RESPONSE RULES:
- Return **only** a valid JSON array of objects.
- Use **double quotes** for all keys and string values.
- No markdown, headings, or extraneous text.
- No semicolons (`;`). No trailing commas.
- Maintain consistent mappings for duplicate errors.
```

Listagem 34: Mapeamento CWE - Teste 6 *prompt* primeiro pedido instrução *user*

```
### Error List:
+ "\n".join(f"{k}: {v}" for k, v in chunk.items())
+ "\n\n### Output format:
  Return exactly a JSON array where each object has these keys, in this order:
    sast_error, cwe, cwe_description, solutions
  - `sast_error`: the error ID string
  - `cwe`: official CWE identifier (e.g., \"CWE-79\")
  - `cwe_description`: official CWE title
  - `solutions`: array of 1-2 solution strings
  No extra keys or text."
```

Listagem 35: Mapeamento CWE - Teste 6 *prompt* segundo pedido instrução *system* - parte 1

```
You are a senior security analyst with deep expertise in application
security and the MITRE CWE taxonomy.
For each SAST finding below, re-evaluate the initial CWE mapping in
context of the raw error text, and then output only a JSON array of
objects, where each object has exactly these keys in this order:
  1. sast_error
  2. verified
  3. cwe
  4. cwe_description
  5. justification
  6. solutions
  7. reasoning

Rules for the JSON:
- Use double quotes only; no single quotes, no semicolons, no markdown, no
extra fields.
- `verified`: true if the initial mapping is correct; false otherwise.

If `verified` is true:
- Keep `cwe` and `cwe_description` exactly as provided.
- You MAY leave `justification` as `""`.
- **Re-emit** the original `solutions` array unchanged.
If `verified` is false:
1. **Replace** `cwe` with a different, correct CWE identifier (it must not
↪  equal the initial value).
2. **Replace** `cwe_description` with that CWE's official title.
3. Provide a concise `justification` (1-2 sentences) explaining why the
↪  original mapping was incorrect.
4. Provide at least 2 *brand-new* `solutions` that you have *not*
↪  mentioned in the first stage.

Example (initial was CWE-494, correct is CWE-497):
{
  "sast_error":"EI_EXPOSE_REP",
  "verified":false,
  "cwe":"CWE-497",
  "cwe_description":"Exposure of Internal Representation",
  "justification":"The error describes exposing internal representation
  ↪  rather than download integrity, so CWE-497 is correct.",
  "solutions":["Return a defensive copy instead of the original
  reference.","Use immutable objects or encapsulation to prevent external
  state leakage."],
  "reasoning":"- The initial mapping to CWE-494 is wrong because that CWE
  concerns download integrity.\n- The error text refers to exposing
  mutable internal state, matching CWE-497.\n- Therefore, the mapping
  must change from 494 to 497 and solutions must target representation
  exposure."
}
```

Listagem 36: Mapeamento CWE - Teste 6 *prompt* segundo pedido instrução *system* - parte 2

```
`reasoning`: a multi-line string capturing your full chain-of-thought for
↪  each mapping.

If you absolutely cannot determine correctness, output exactly:
{
  "sast_error":"<ERROR_ID>",
  "verified":false,
  "cwe":"",
  "cwe_description":"",
  "justification":"CWE mapping unclear. Needs manual review.",
  "solutions":[],
  "reasoning":"Unable to match the error text to a CWE with confidence;
  ↪  manual review required."
}
```

A.6 TESTE 7

Listagem 37: Mapeamento CWE - Teste 7 *prompt* primeiro pedido instrução *system*

```
You are an expert in application security and vulnerability analysis with
↪  deep knowledge
of MITRE CWE classifications.
For each SAST error, identify the most accurate CWE identifier and its
↪  official description,
and recommend 1-2 concise, industry-standard solutions.
### RESPONSE RULES:
- Return **only** a valid JSON array of objects.
- Use **double quotes** for all keys and string values.
- No markdown, headings, or extraneous text.
- No semicolons (`;`). No trailing commas.
- Maintain consistent mappings for duplicate errors.
```

Listagem 38: Mapeamento CWE - Teste 7 *prompt* primeiro pedido instrução *user*

```
### Error List:
+ "\n".join(f"{k}: {v}" for k, v in chunk.items())
+ "\n\n### Output format:
  Return exactly a JSON array where each object has these keys, in this
  ↪  order:
    sast_error, cwe, cwe_description, solutions
  - `sast_error`: the error ID string
  - `cwe`: official CWE identifier (e.g., \"CWE-79\")
  - `cwe_description`: official CWE title
  - `solutions`: array of 1-2 solution strings
  No extra keys or text."
```

Listagem 39: Mapeamento CWE - Teste 7 *prompt* segundo pedido instrução *system* - parte 1

```
You are a senior security analyst with deep expertise in application security and the MITRE CWE taxonomy.
For each SAST finding below, re-evaluate the initial CWE mapping in context of the raw error text,
and then output only a JSON array of objects, where each object has exactly these keys in this order:
  1. sast_error
  2. verified
  3. cwe
  4. cwe_description
  5. justification
  6. solutions
  7. reasoning

Rules for the JSON:
- Use double quotes only; no single quotes, no semicolons, no markdown, no extra fields.
- `verified`: true if the initial mapping is correct; false otherwise.

If `verified` is true:
- Keep `cwe` and `cwe_description` exactly as provided.
- You MAY leave `justification` as `""`.
- **Re-emit** the original `solutions` array unchanged.

If `verified` is false:
1. **Replace** `cwe` with a different, correct CWE identifier (it must not equal the initial value).
2. **Replace** `cwe_description` with that CWE's official title.
3. Provide a concise `justification` (1-2 sentences) explaining why the original mapping was incorrect.
4. Provide at least 2 *brand-new* `solutions` that you have *not* mentioned in the first stage.

Example (initial was CWE-494, correct is CWE-497):
{
  "sast_error":"EI_EXPOSE_REP",
  "verified":false,
  "cwe":"CWE-497",
  "cwe_description":"Exposure of Internal Representation",
  "justification":"The error describes exposing internal representation rather than download integrity, so CWE-497 is correct.",
  "solutions":[
    "Return a defensive copy instead of the original reference.",
    "Use immutable objects or encapsulation to prevent external state leakage."
  ],
  "reasoning":"- The initial mapping to CWE-494 is wrong because that CWE concerns download integrity.\n- The error text refers to exposing mutable internal state, matching CWE-497.\n- Therefore, the mapping must change from 494 to 497 and solutions must target representation exposure."
}
```

Listagem 40: Mapeamento CWE - Teste 7 *prompt* segundo pedido instrução *system* - parte 2

```
`reasoning`: a multi-line string capturing your full chain-of-thought for each mapping.

If you absolutely cannot determine correctness, output exactly:
{
  "sast_error":"<ERROR_ID>",
  "verified":false,
  "cwe":"",
  "cwe_description":"",
  "justification":"CWE mapping unclear. Needs manual review.",
  "solutions":[],
  "reasoning":"Unable to match the error text to a CWE with confidence; manual review required."
}
```

Listagem 41: Mapeamento CWE - Teste 7 *prompt* terceiro pedido instrução *system*

```
You are a senior security analyst. For each SAST finding below, perform a final re-evaluation using:
1. The raw error text.
2. The initial CWE mapping (Pass 1).
3. The verified CWE mapping, justification, and solutions (Pass 2).

Your task:
- Decide whether the verified mapping should stand or be updated.
- Choose `final_cwe` (e.g. "CWE-497").
- **Derive** `final_description` **only** from that CWE's official title. Do **not** invent, reuse, or copy an unrelated description.
- Provide at least **2 brand-new** `final_solutions` specific to the `final_cwe`.
- Provide a concise `justification` (1-2 sentences) explaining any change or confirming correctness.
- Provide a full chain-of-thought in `reasoning`.
- **Consistency rule**: At the end of your `reasoning`, include the line:
    → Confirmed mapping: CWE-XXX (Official Title)
  where `CWE-XXX` and `(Official Title)` exactly match your `final_cwe` and `final_description`.

Output **only** a JSON array of objects, each with exactly these keys in this order:
  1. `sast_error`
  2. `final_cwe`
  3. `final_description`
  4. `final_solutions`
  5. `justification`
  6. `reasoning`

Rules:
- Use double quotes only; no single quotes, semicolons, markdown, or extra fields.
- Before returning, internally verify that `final_description` exactly matches the official title for `final_cwe`. If they do not match, correct them.
```

A.7 TESTE 8

Listagem 42: Mapeamento CWE - Teste 8 *prompt* primeiro pedido

```
You are an expert in application security and vulnerability analysis, with deep knowledge of CWE classifications following OWASP and MITRE standards.
Below is a list of SAST tool errors. Each error includes an identifier and a tool-generated description.

For each error:
- Identify the **single most accurate CWE** based on MITRE's official taxonomy.
- Provide a **clear, standardized CWE description**.
- Recommend **one or two concise, industry-standard solutions**.

### Error List:
{chr(10).join([f"{k}: {v}" for k, v in chunk.items()])}

### Response Format:
Return a **valid JSON array**, with each object using this structure:
[
  {{
    "sast_error": "<ERROR_ID>",
    "cwe": "<CWE_IDENTIFIER>",
    "cwe_description": "<CWE_DESCRIPTION>",
    "solutions": [
      "<SOLUTION_1>",
      "<SOLUTION_2>"
    ]
  }},
]

### Output Requirements:
- **Return a JSON array with one object per error.**
- **If no CWE applies, set `"cwe": ""`, `"cwe_description": ""`**
- **Use the exact same CWE for the same error every time.** Prioritize widely accepted mappings.
- **Do not guess or infer CWE mappings.** Follow MITRE strictly.
- **Avoid subjective or loosely related CWEs.**
- Ensure the output is **valid JSON**:
  - No additional text or formatting (e.g., markdown).
  - No semicolons (`;`).
  - No trailing commas.

- Keep descriptions and solutions **uniform, professional, and phrased with industry terminology**.
```

Listagem 43: Mapeamento CWE - Teste 8 *prompt* segundo pedido

```
You are a security analyst. Review the original list of errors, and then my AI's initial (possibly empty) CWE mappings.
Original Errors:
{chr(10).join([f"{k}: {v}" for k, v in chunk.items()])}
Initial Mappings:
{mapping_entries}

For each entry:
- If the CWE is correct, return the same CWE and mark "verified": true.
- If incorrect or missing:
  - Set "verified": false.
  - Provide the correct "cwe", its "cwe_description", and a brief "justification".
  - Also provide one or two corrected, industry-standard "solutions" for the corrected CWE.

Return one object for every original error, regardless of whether it was in the initial mappings.
{{
  "sast_error": "<ERROR_ID>",
  "verified": true/false,
  "cwe": "<CWE_IDENTIFIER>",
  "cwe_description": "<CWE_DESCRIPTION>",
  "justification": "<JUSTIFICATION or empty string>",
  "solutions": ["<SOLUTION_1>", "<SOLUTION_2>"]
}}

### Output Requirements:
- **Do not guess or infer CWE mappings.** Follow MITRE strictly.
- **Avoid subjective or loosely related CWEs.**
- Ensure the output is **valid JSON**:
  - No additional text or formatting (e.g., markdown).
  - No semicolons (`;`).
  - No trailing commas.

- Keep descriptions and solutions **uniform, professional, and phrased with industry terminology**.
```

B

APÊNDICE B - CLASSIFICAÇÕES DOS TESTES DA PLATAFORMA

Tabela B.1: CWE-23 Resultados por limiar de probabilidade

Limiar	**TP**	**TP_mislabel**	**FP**	**FN**
50	238	51	3010	405
60	238	51	2973	405
70	238	33	2173	405
80	238	18	807	405
90	8	0	1	405

Tabela B.2: CWE-23 Resultados por limiar de pontuação de risco

Limiar	**TP**	**TP_mislabel**	**FP**	**FN**
50	238	16	392	405
60	238	16	392	405
70	238	16	392	405
80	44	0	174	405
90	6	0	6	405

Tabela B.3: CWE-23 Resultados por limiar de SQI

Limiar	**TP**	**TP_mislabel**	**FP**	**FN**
5	238	16	392	405
6	45	0	177	405
7	14	0	7	405
8	0	0	0	405
9	0	0	0	405

Tabela B.4: CWE-36 Resultados por limiar de probabilidade

Limiar	**TP**	**TP_mislabel**	**FP**	**FN**
50	243	58	2058	405
60	243	58	2056	405
70	243	28	1322	405
80	243	16	702	405
90	18	0	0	405

Tabela B.5: CWE-36 Resultados por limiar de pontuação de risco

Limiar	**TP**	**TP_mislabel**	**FP**	**FN**
50	243	16	393	405
60	243	16	393	405
70	243	16	392	405
80	150	6	256	405
90	2	0	27	405

Tabela B.6: CWE-36 Resultados por limiar de SQI

Limiar	**TP**	**TP_mislabel**	**FP**	**FN**
5	243	16	393	405
6	151	16	284	405
7	20	0	27	405
8	0	0	0	405
9	0	0	0	405

Tabela B.7: CWE-78 Resultados por limiar de probabilidade

Limiar	**TP**	**TP_mislabel**	**FP**	**FN**
50	286	67	2267	403
60	286	67	2239	403
70	286	46	1809	403
80	286	19	400	403
90	100	0	37	403

Tabela B.8: CWE-78 Resultados por limiar de pontuação de risco

Limiar	**TP**	**TP_mislabel**	**FP**	**FN**
50	286	16	304	403
60	286	16	304	403
70	286	13	298	403
80	286	0	246	403
90	187	0	185	403

Tabela B.9: CWE-78 Resultados por limiar de SQI

Limiar	**TP**	**TP_mislabel**	**FP**	**FN**
5	286	16	304	403
6	286	0	246	403
7	286	0	221	403
8	1	0	1	403
9	0	0	0	403

Tabela B.10: CWE-89 Resultados por limiar de probabilidade

Limiar	**TP**	**TP_mislabel**	**FP**	**FN**
50	3280	344	18689	1876
60	3280	344	18529	1876
70	3280	196	14816	1876
80	3280	123	11175	1876
90	2403	0	6629	1876

Tabela B.11: CWE-89 Resultados por limiar de pontuação de risco

Limiar	**TP**	**TP_mislabel**	**FP**	**FN**
50	3280	75	10451	1876
60	3280	75	10451	1876
70	3280	75	10440	1876
80	3280	0	9890	1876
90	1698	0	5207	1876

Tabela B.12: CWE-89 Resultados por limiar de SQI

Limiar	TP	TP__mislabel	FP	FN
5	3280	75	10450	1876
6	3280	0	10072	1876
7	3251	0	9447	1876
8	850	0	2389	1876
9	0	0	0	1876

Tabela B.13: CWE-134 Resultados por limiar de probabilidade

Limiar	TP	TP__mislabel	FP	FN
50	280	96	3533	644
60	280	96	3531	644
70	280	74	2454	644
80	280	58	1496	644
90	0	0	1	644

Tabela B.14: CWE-134 Resultados por limiar de pontuação de risco

Limiar	TP	TP__mislabel	FP	FN
50	280	30	579	644
60	280	30	579	644
70	280	30	577	644
80	0	23	307	644
90	0	0	54	644

Tabela B.15: CWE-134 Resultados por limiar de SQI

Limiar	TP	TP__mislabel	FP	FN
5	280	30	578	644
6	0	30	384	644
7	0	0	55	644
8	0	0	0	644
9	0	0	0	644

Tabela B.16: CWE-190 Resultados por limiar de probabilidade

Limiar	**TP**	**TP_mislabel**	**FP**	**FN**
50	0	1442	23129	5060
60	0	1442	23108	5060
70	0	1093	12881	5060
80	0	234	4067	5060
90	0	0	0	5060

Tabela B.17: CWE-190 Resultados por limiar de pontuação de risco

Limiar	**TP**	**TP_mislabel**	**FP**	**FN**
50	0	639	2085	5060
60	0	639	2085	5060
70	0	77	1168	5060
80	0	30	561	5060
90	0	0	168	5060

Tabela B.18: CWE-190 Resultados por limiar de SQI

Limiar	**TP**	**TP_mislabel**	**FP**	**FN**
5	0	79	1210	5060
6	0	69	695	5060
7	0	0	168	5060
8	0	0	0	5060
9	0	0	0	5060

DECLARAÇÃO

Declaro, sob compromisso de honra, que o trabalho apresentado nete projeto, com o título *"VulnFusion: Plataforma de Integração de Ferramentas Open-Source SAST"*, é original e foi realizado por Edgar Emanuel Raimundo Andrade (2230054) sob orientação de Professor Doutor Paulo Jorge Gonçalves Loureiro (paulo.loureiro@ipleiria.pt).

Leiria, Setembro de 2025

Edgar Emanuel Raimundo Andrade