Detecção Direta versus Indireta em Pipelines DevSecOps: Análise de Ruído Taxonômico e Calibração de Portões de Segurança Progressivos

Guilherme Machado[1], **Fábio Cordeiro**[1], **Caroline Jandre**[1]

[1]Departamento de Sistemas de Informação
Pontifícia Universidade Católica de Minas Gerais - Belo Horizonte - MG - Brasil

guilhermeoh.machado@gmail.com, fabioleandro@pucminas.br, carolinejandre@pucminas.br

Abstract. *The aggregation of findings in multilayer DevSecOps pipelines relies on common taxonomies such as CWE and CVSS, which obscure an operational asymmetry: a given weakness category may be reported both as an application-level flaw identified directly in source code and as a CVE detected in stack components whose CWE coincides with that category. Treating both as equivalent inflates coverage metrics and contributes to alert fatigue. This work formalizes the distinction between direct and indirect detections and characterizes its prevalence in a six-layer pipeline against a controlled web target. We then derive a model of progressive security gates that incorporates this distinction and evaluate three gate policies through simulation. Results show the direct/indirect split materially affects gate behavior and supports calibration strategies that preserve detection reach while avoiding spurious build interruptions.*

Resumo. *A agregação de achados em pipelines DevSecOps multicamada apoia-se em taxonomias comuns como CWE e CVSS, que ocultam uma assimetria operacional: uma mesma categoria de fraqueza pode ser reportada tanto como falha aplicacional identificada diretamente no código-fonte, quanto como CVE detectada em componentes da stack cujo CWE coincide com essa categoria. Tratar ambas como equivalentes inflaciona métricas de cobertura e contribui para a fadiga de alertas. Este trabalho formaliza a distinção entre detecções diretas e indiretas e caracteriza sua prevalência em um pipeline de seis camadas executado contra uma aplicação web alvo controlada. A partir dessa caracterização, deriva-se um modelo de portões de segurança progressivos que incorpora tal distinção, avaliado por simulação de três políticas. Os resultados mostram que a separação direto/indireto afeta materialmente o comportamento dos portões e fundamenta calibrações que preservam o alcance da detecção evitando interrupções espúrias do build.*

1. Introdução

A integração contínua e a entrega contínua (*Continuous Integration/Continuous Delivery*, CI/CD) tornaram-se alicerces da engenharia de *software* moderna [Humble and Farley 2010, Forsgren et al. 2018]. A cultura *DevSecOps* estende esse modelo ao incorporar verificações automatizadas de segurança ao longo do *pipeline*, materializando o princípio de *shift-left*: mover a detecção de falhas para

fases iniciais do ciclo de vida para reduzir custos de correção e dívida técnica de segurança [Chen and Suo 2022, Kushwaha et al. 2024, Feio et al. 2024].

Em *pipelines* modernos, essa abrangência é obtida pela orquestração de múltiplas camadas de análise, tipicamente SAST (*Static Application Security Testing*), DAST (*Dynamic Application Security Testing*), SCA (*Software Composition Analysis*), *Infrastructure as Code Scan* e *Container Scan*, complementadas eventualmente por testes customizados [Masood and Java 2015, Rangnau et al. 2020, Bernardino et al. 2024]. Cada camada opera sobre um artefato distinto (código-fonte, dependências, imagens, configuração de infraestrutura, aplicação em execução) e emite achados categorizados por identificadores comuns como CVE (*Common Vulnerabilities and Exposures*), CWE (*Common Weakness Enumeration*) e CVSS (*Common Vulnerability Scoring System*). Tais identificadores são usados tanto para agregação quanto para decisão automatizada: políticas de *quality gates* filtram achados por severidade ou classe e interrompem o *build* quando limites são excedidos [Marandi et al. 2023, Putra and Kabetta 2022].

Essa uniformização taxonômica, contudo, obscurece uma distinção operacional relevante: um mesmo CWE pode ser reportado por ferramentas que operam sobre artefatos fundamentalmente diferentes. O SAST identifica, por exemplo, CWE-89 (*SQL Injection*) como padrão inseguro no código-fonte; o *Container Scan*, por sua vez, reporta CWE-89 em CVEs de bibliotecas do sistema operacional que expõem a *stack* a injeções. Ambos compartilham a mesma classificação taxonômica, mas representam riscos distintos: o primeiro é uma falha aplicacional demonstrável; o segundo, um indicador de superfície de ataque na infraestrutura hospedeira. Tratar ambos como equivalentes em métricas de cobertura ou políticas de portão inflaciona a confiança no *pipeline* e contribui para a fadiga de alertas reportada como obstáculo recorrente da adoção de DevSecOps [Rajapakse et al. 2022]. Na prática, a classificação é operacionalizada pelo confronto entre dois elementos declarados *a priori*: o artefato-alvo de cada ferramenta e uma matriz de vulnerabilidades esperadas da aplicação. Um achado é direto quando a ferramenta reporta a fraqueza no mesmo artefato em que a vulnerabilidade da matriz reside; é indireto quando a correspondência se dá apenas pela coincidência do identificador CWE em artefato distinto. Coloca-se, então, a seguinte pergunta de pesquisa: como distinguir formalmente risco aplicacional de risco de *stack* em *pipelines* DevSecOps multicamada, e como essa distinção pode calibrar portões de segurança progressivos que equilibrem cobertura e ação bloqueante?

Diante do exposto, este trabalho tem como objetivo formalizar a distinção entre detecções diretas e indiretas em *pipelines* DevSecOps multicamada e avaliar empiricamente seu uso na calibração de portões de segurança progressivos. O experimento foi conduzido sobre a *Google Cloud Platform*, integrando *Semgrep*, *Trivy*, *Checkov*, *OWASP ZAP (Open Web Application Security Project Zed Attack Proxy)* e um módulo *Python* de teste de credenciais, com a *Damn Vulnerable Web Application* (DVWA) como aplicação-alvo. O *pipeline* coletou 1.750 ocorrências brutas de vulnerabilidades, sobre as quais foram simuladas três políticas de portão para mensurar o efeito da distinção direto/indireto.

A contribuição consiste em: (i) um modelo de classificação de detecções que separa achados diretos, em que a ferramenta identifica a falha no artefato-alvo que é sua unidade de análise nativa, de achados indiretos, em que a ferramenta identifica CVE em componente da *stack* cujo CWE coincide com uma vulnerabilidade prevista da aplicação;

(ii) uma caracterização empírica do ruído taxonômico em um *pipeline* de seis camadas executado sobre a DVWA; (iii) um modelo de portões de segurança progressivos (*Commit* $\rightarrow$ *Build* $\rightarrow$ *Deploy* $\rightarrow$ *Runtime*) calibrado pela distinção direto/indireto, avaliado por simulação de três políticas sobre os dados coletados; (iv) um pacote de artefatos reprodutível com os relatórios JSON brutos, o *script* de consolidação e a matriz de referência[1].

Esta proposta contribui para a área de segurança ao oferecer um critério adicional de calibração para *pipelines* DevSecOps existentes, ortogonal a técnicas de priorização baseadas em severidade ou *exploitability*. A avaliação empírica evidencia concentração expressiva de detecções indiretas no *Container Scan* sobre a imagem base *Debian 9.5* (EOSL - *End of Support Life*), e mostra que tratar essas detecções como sinal informativo, em vez de bloqueante, reduz substancialmente a taxa de bloqueios espúrios.

O restante do artigo apresenta a fundamentação teórica (Seção 2), os trabalhos relacionados (Seção 3), o modelo proposto (Seção 4), a metodologia experimental (Seção 5), os resultados, ameaças à validade e discussão (Seção 6) e as conclusões com apontamentos de trabalho futuro (Seção 7).

2. Referencial Teórico

2.1. DevSecOps e Shift-Left

DevSecOps estende a cultura DevOps ao materializar a segurança através do paradigma de *Security as Code*: políticas e controles deixam de ser processos manuais para se tornarem artefatos de *software* versionáveis e automatizáveis [Chen and Suo 2022, Ahmed and Francis 2019, Feio et al. 2024, Efendi et al. 2021]. A estratégia de *shift-left* operacionaliza esse princípio ao introduzir verificações automatizadas nas etapas iniciais do *pipeline*: vulnerabilidades são detectadas enquanto o contexto técnico é recente para o desenvolvedor, minimizando o custo de correção e evitando o acúmulo de dívida técnica de segurança [Humble and Farley 2010, Kushwaha et al. 2024].

2.2. Teste Contínuo de Segurança e Taxonomias

O Teste Contínuo de Segurança define a aplicação de testes automatizados de forma ininterrupta ao longo do *pipeline* [Feio et al. 2024, Bernardino et al. 2024]. A literatura converge em cinco categorias canônicas: SAST, análise estática do código-fonte buscando padrões inseguros sem execução [Masood and Java 2015]; DAST, teste dinâmico que interage com a aplicação em *runtime* simulando ataques externos [Rangnau et al. 2020]; SCA, identificação de vulnerabilidades conhecidas em bibliotecas e dependências de terceiros; IaC Scan, varredura de arquivos de Infraestrutura como Código para detectar configurações inseguras; e *Container Scan*, varredura de imagens para identificar CVEs no sistema operacional base, pacotes instalados e sinalização de imagens em *End of Support Life* (EOSL). Pipelines podem ainda incorporar testes customizados para vetores específicos não cobertos pelas cinco categorias acima, por exemplo, testes ativos de força bruta com tratamento dinâmico de mecanismos anti-CSRF (*Cross-Site Request Forgery*).

A classificação dos achados baseia-se em padrões globais de segurança. A CWE [MITRE 2026b] categoriza tipos de fraquezas de *software*, servindo como linguagem comum para descrever falhas. CVEs são instâncias específicas dessas fraquezas

[1] `https://github.com/Machada1/sbseg2026-deteccao-direta-indireta`

em produtos concretos, rastreadas no *National Vulnerability Database* [MITRE 2026a, NIST 2026]. A severidade é quantificada pelo CVSS [FIRST 2026], que atribui pontuação de 0 a 10 com base em impacto e facilidade de exploração. Esse vocabulário comum viabiliza a agregação de achados heterogêneos, mas também origina a ambiguidade explorada neste trabalho.

Para tratar dessa ambiguidade de forma operacional, dois conceitos preliminares são úteis. O primeiro é o de *artefato-alvo*: a unidade de análise nativa sobre a qual uma ferramenta de segurança opera. Ferramentas SAST têm como artefato-alvo o código-fonte; DAST tem a aplicação em execução; SCA tem o manifesto de dependências; o *Container Scan* tem a imagem construída; e o IaC Scan tem os arquivos de infraestrutura como código. O segundo é o de *vulnerabilidade esperada*: a vulnerabilidade está associada a um artefato específico em que ela foi implementada, o código-fonte para vulnerabilidades aplicacionais como *SQL Injection*, e a imagem de contêiner para vulnerabilidades de infraestrutura como uso de sistema operacional em fim de suporte. É a relação entre o artefato-alvo da ferramenta e o artefato em que a vulnerabilidade reside que fundamenta a distinção formalizada na Seção 4 e empregada em todo o trabalho.

2.3. Portões de Segurança e Fadiga de Alertas

Quality gates ou, no contexto de segurança, *security gates*, são mecanismos automatizados que condicionam o avanço do *pipeline* à satisfação de critérios predefinidos sobre os achados coletados, tipicamente *thresholds* sobre severidade ou contagens por classe [Marandi et al. 2023, Putra and Kabetta 2022]. A calibração desses portões é dificultada pela fadiga de alertas: o volume de achados gerados por um *pipeline* multicamada frequentemente excede a capacidade de triagem da equipe, levando ao descarte indiscriminado ou ao relaxamento dos *thresholds* até um ponto em que o portão perde função prática, condição reconhecida como obstáculo recorrente da adoção de DevSecOps em revisão sistemática recente [Rajapakse et al. 2022]. A resposta canônica é a priorização: ordenar achados por combinações de CVSS, *exploitability*, criticidade do ativo e proximidade à produção [Ribeiro et al. 2024]. Este trabalho incorpora à calibração uma dimensão complementar a essas combinações, a classe de detecção, formalizada na Seção 4.

3. Trabalhos Relacionados

A literatura distribui-se por dois eixos principais para este trabalho: a orquestração de *pipelines* DevSecOps multicamada e a análise e consolidação de achados de segurança no contexto de fadiga de alertas.

3.1. Orquestração de Pipelines DevSecOps

Frameworks conceituais estabelecem a base para integração de práticas de segurança em CI/CD, com estudos de caso integrando SAST, DAST e SCA [Feio et al. 2024], arquiteturas focadas em automação [Chen and Suo 2022] e modelos de transformação cultural a partir do *Waterfall* [Efendi et al. 2021]. Estudos empíricos demonstram a viabilidade de orquestrar SAST e DAST em *pipelines* sobre *GitLab*/*Docker* [Putra and Kabetta 2022], *GitHub Actions* [Marandi et al. 2023] e *Azure DevOps* [Kushwaha et al. 2024], bem como aprofundam desafios específicos: o *overhead* e os desafios de integração do DAST autenticado com OWASP ZAP em CI/CD [Rangnau et al. 2020], a complementaridade entre análise *white-box* e *black-box* [Masood and Java 2015], e a varredura de IaC como

camada crítica de provisionamento [Bernardino et al. 2024]. Projetos de arquitetura buscam unificar resultados em plataformas centralizadas [Sun et al. 2021], sustentados por orquestradores de contêineres [Díaz et al. 2019].

A presente pesquisa adota os modelos de [Feio et al. 2024] e [Chen and Suo 2022] como fundação estrutural. O diferencial do trabalho está no tratamento analítico dos achados produzidos: onde a literatura tipicamente mensura cobertura por agregação taxonômica, este trabalho propõe uma dimensão adicional, a classe de detecção, e avalia seu efeito sobre a calibração de portões.

3.2. Análise de Achados e Calibração de Portões

A heterogeneidade de esquemas de reporte entre ferramentas é um problema reconhecido. Soluções centralizadas de gerenciamento, agrupadas sob o termo *Application Security Posture Management* (ASPM), propõem normalizar achados em um formato comum para priorização [Sun et al. 2021]. Trabalhos recentes propõem abordagens de priorização baseadas em aprendizado de máquina [Ribeiro et al. 2024], contexto de exploração [Ponce et al. 2024] e comportamento de desenvolvedores [Sousa et al. 2024]. A maioria opera sobre um único eixo de calibração, geralmente severidade ou probabilidade de exploração; poucos trabalhos consideram explicitamente a posição do portão no ciclo de vida da aplicação como variável de projeto.

No contexto acadêmico brasileiro, trabalhos recentes do SBSeg abordam problemas afins. [Ribeiro et al. 2024] empregam processos gaussianos e aprendizado ativo para classificar risco de vulnerabilidades em larga escala. [Ponce et al. 2024] enriquecem a análise de vulnerabilidades por meio de *fingerprinting* ativo de serviços expostos na Internet, aumentando em até 14 vezes as informações disponíveis em relação a motores de busca como o Shodan. [Sousa et al. 2024] analisam comportamentos de desenvolvedores na mitigação de vulnerabilidades em projetos *open-source* e [Kujavski et al. 2024] caracterizam o uso de *software* desatualizado em produção, evidenciando que imagens e pacotes em fim de suporte são vetores de risco persistentes, fenômeno observado em nossos resultados sobre o *Container Scan*.

Ainda no SBSeg, [Sacramento et al. 2024] abordam o problema específico de automação de autenticação em testes com OWASP ZAP para contornar formulários com proteção CSRF. Esse problema é parente do que motivou, neste trabalho, o desenvolvimento de um módulo *Python* de teste de força bruta com gestão dinâmica de *tokens*. A contribuição do presente artigo, contudo, não reside nesse módulo, que é tratado como etapa de coleta cujas diferenças operacionais são documentadas na Seção 5.

Este estudo diferencia-se dos trabalhos citados por uma mudança de foco: em vez de propor uma nova técnica de priorização ou uma ferramenta adicional, propõe-se uma distinção taxonômica que opera sobre os achados já emitidos pelas ferramentas existentes. A formalização direto/indireto é ortogonal às técnicas de priorização da literatura e pode ser composta com elas, e o modelo de portões progressivos endereça a posição do portão no ciclo de vida como variável explícita de projeto.

4. Modelo Proposto

4.1. Classes de Detecção: Direta e Indireta

A partir dos conceitos de artefato-alvo e vulnerabilidade esperada introduzidos na Seção 2.2, define-se a classe de detecção pela relação entre ambos. Neste trabalho, é considerada uma detecção direta quando a ferramenta identifica a vulnerabilidade no mesmo artefato em que ela foi implementada. No caso de vulnerabilidades de infraestrutura, considera-se como artefato da vulnerabilidade a própria imagem, pacote ou configuração analisada, de modo que achados de *Container Scan* ou IaC Scan sobre esses elementos também constituem detecções diretas. Uma detecção é indireta quando a ferramenta identifica um sintoma em artefato diferente, associando-o à vulnerabilidade por meio de coincidência taxonômica na classificação CWE.

A definição é operacional: aplicada a um *pipeline* concreto e a uma matriz de vulnerabilidades esperadas, ela produz um mapeamento explícito entre cada par (ferramenta, vulnerabilidade) e sua classe de detecção. A classe é uma propriedade do par (ferramenta, vulnerabilidade) num contexto específico de *pipeline*: a mesma ferramenta pode ser direta em relação a uma vulnerabilidade e indireta em relação a outra. Uma detecção indireta não é um falso positivo: o CWE reportado é legitimamente presente no componente; o que difere é o escopo de impacto, pois o componente da *stack* pode nunca ser invocado em um caminho exploratório da aplicação. Sob a perspectiva de defesa em profundidade, detecções indiretas têm valor informativo, e a recomendação operacional não é ignorá-las, mas tratá-las diferentemente na decisão de bloqueio.

A classificação proposta é deliberadamente estrutural: depende do par (ferramenta, vulnerabilidade), e não de características intrínsecas de cada achado, como severidade ou alcançabilidade do componente. O benefício é o determinismo, pois dada a matriz de referência e a declaração dos artefatos-alvo, a classificação é reprodutível sem julgamento humano por achado; o custo é tratar uniformemente achados indiretos heterogêneos, e incorporar características dos achados como refinamento intraclasse fica como trabalho futuro. Quando uma CVE está associada a múltiplos CWEs (15,6% das CVEs do *Container Scan*), a correspondência considera todos os identificadores; o casamento é por identificador exato, sem resolução da hierarquia CWE, escolha conservadora que pode omitir correspondências legítimas, mas não produz correspondências espúrias. Ambas as decisões são auditáveis no pacote de artefatos. A Figura 1 ilustra a distinção com um exemplo do experimento.

4.2. Portões de Segurança Progressivos

O modelo organiza o ciclo de vida da aplicação em quatro estágios sequenciais que correspondem às fases canônicas de *pipelines* CI/CD [Humble and Farley 2010], instrumentadas para decisões de segurança automatizadas. A progressão reflete o princípio de que o rigor da decisão deve escalar com a proximidade do artefato à produção, e que cada estágio tem informação de natureza diferente.

O *Commit Gate* opera sobre o código-fonte recém-submetido e é alimentado exclusivamente por SAST e SCA, cujos achados possuem relação mais direta com vulnerabilidades aplicacionais ou dependências explicitamente utilizadas pela aplicação. A ação padrão é bloqueante para achados diretos de severidade alta e crítica.

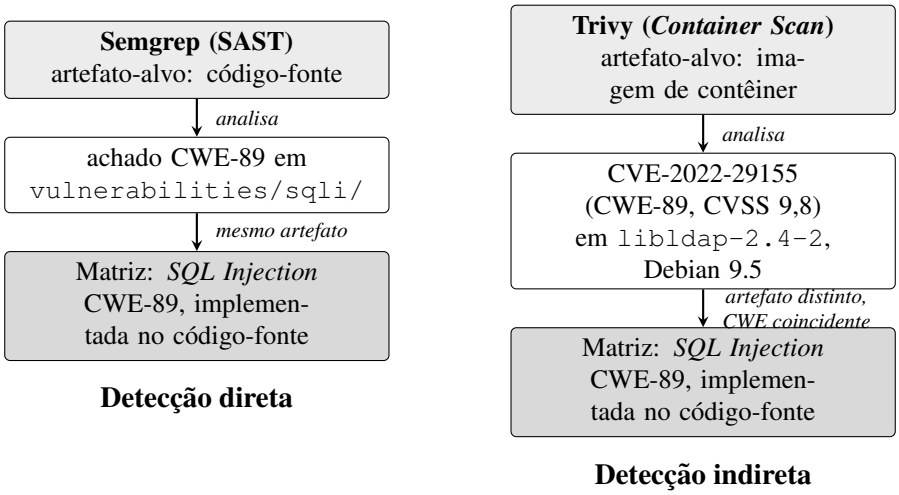

Figura 1. Detecção direta e indireta para a mesma vulnerabilidade da matriz (*SQL Injection*, CWE-89): o Semgrep reporta a fraqueza no artefato em que foi implementada; o *Container Scan* reporta a CVE-2022-29155 em pacote do Sistema Operacional da imagem, coincidindo apenas no identificador CWE.

O *Build Gate* opera sobre a imagem de contêiner e a infraestrutura empacotada, alimentado por *Container Scan* e IaC Scan. Esta é a etapa em que a maioria das detecções indiretas se manifesta. O modelo prescreve que o *Build Gate* bloqueie o avanço para achados diretos (vulnerabilidades nas próprias camadas de infraestrutura, como EOSL ou *Network Policies* ausentes) e registre, sem bloquear, achados indiretos. Esta é a decisão-chave que o modelo proposto introduz.

O *Deploy Gate* opera sobre a aplicação já instanciada em ambiente de homologação, alimentado por DAST e testes customizados. Achados nessas camadas demonstram exploração efetiva em *runtime* e são diretos em relação a vulnerabilidades aplicacionais; a ação é bloqueante para qualquer achado confirmado por *Active Scan* autenticado ou teste customizado.

O *Runtime Watch* não é um portão bloqueante: é monitoramento contínuo que realimenta os estágios anteriores com base em telemetria e *logs*. Não é demonstrado por simulação neste trabalho, pois o ambiente controlado não gerou tráfego exploratório, e fica como fronteira de trabalho futuro. A Tabela 1 consolida o mapeamento entre camadas, estágios e ações padrão.

Tabela 1. Estágios de portão, camadas alimentadoras e ações padrão do modelo proposto.

Estágio	Camadas alimentadoras	Ação padrão
Commit Gate	SAST, SCA	bloqueio em direto Alto/Crítico
Build Gate	Container Scan, IaC Scan	bloqueio em direto; informativo em indireto
Deploy Gate	DAST, testes customizados	bloqueio em exploração confirmada
Runtime Watch	telemetria, logs	realimentação (não bloqueante)

4.3. Políticas de Calibração

A calibração é expressa por políticas que parametrizam o *threshold* de severidade e o tratamento dado a detecções indiretas. Definimos três políticas que cobrem o espectro

entre permissividade e rigor. A política P1 (Permissiva) bloqueia apenas achados de severidade CVSS $\geq 9{,}0$ em qualquer estágio, tratando diretos e indiretos uniformemente, representando a política mais simples, baseada apenas em severidade. A política P2 (Estrita uniforme) bloqueia achados de CVSS $\geq 7{,}0$ em qualquer estágio, sem discriminação por classe. A política P3 (Progressiva com classe de detecção) aplica bloqueio em CVSS $\geq 7{,}0$ apenas para achados diretos, e trata indiretos como registro informativo, implementando assim o modelo proposto. P1 e P2 são os referenciais contra os quais o modelo é comparado na Seção 6, que mede o volume de bloqueios disparados por cada política e a preservação da capacidade do *pipeline* de sinalizar as vulnerabilidades esperadas.

5. Metodologia

A metodologia experimental aplica o modelo da Seção 4 sobre uma aplicação-alvo controlada para gerar evidência empírica das proporções direto/indireto e do efeito das políticas de calibração. A estratégia segue duas etapas: primeiro, instrumenta-se um *pipeline* multicamada para produzir um conjunto de achados representativo da diversidade de camadas e classes previstas pelo modelo; segundo, os achados são processados para permitir a avaliação quantitativa da distinção direto/indireto e a simulação das políticas de portão.

5.1. Ambiente Experimental e Aplicação-Alvo

O ambiente foi construído sobre a *Google Cloud Platform*, com *Cloud Build* como orquestrador e a aplicação-alvo executada em um *cluster Google Kubernetes Engine* (GKE) provisionado por *Terraform*. O *Artifact Registry* gerencia imagens e o *Google Cloud Storage* centraliza os artefatos. O *pipeline* é disparado por *triggers* configurados em um ramo específico e orquestra 15 etapas sequenciais.

A aplicação-alvo é a *Damn Vulnerable Web Application* (DVWA) na imagem oficial `vulnerables/web-dvwa`, configurada em nível de segurança baixo. A DVWA é uma aplicação PHP/MySQL deliberadamente vulnerável, com 17 vulnerabilidades documentadas abrangendo categorias do OWASP Top 10 [Dewhurst 2026, Open Worldwide Application Security Project 2025]. A matriz de referência adotada classifica 13 vulnerabilidades como *web_application* e 4 como *infrastructure*. Duas dessas quatro vulnerabilidades de infraestrutura, *Default Credentials* e *Exposed MySQL*, manifestam-se por meio da aplicação em execução e são, portanto, tratadas com camada de detecção `APP` na classificação; as duas restantes, *Outdated OS* e *Outdated Packages*, permanecem na camada `INFRA` com correspondência agregada. A escolha da DVWA decorre da disponibilidade de uma matriz de vulnerabilidades conhecidas *a priori*, condição necessária para construir a classificação direto/indireto de forma rigorosa.

5.2. Ferramental

O instrumental cobre as cinco categorias canônicas listadas na Seção 2.2, complementadas por um teste customizado de credenciais: SAST com *Semgrep*, sobre código PHP e *JavaScript*; SCA e *Container Scan* com *Trivy* (*Trivy FS* sobre dependências e *Trivy Container* sobre a imagem); IaC Scan com *Checkov* sobre *Terraform* e manifestos *Kubernetes*; e DAST com *OWASP ZAP* em duas fases, *Baseline Scan* passiva e *Active Scan* autenticada com a política padrão da ferramenta em integrações CI/CD. O teste customizado é um módulo *Python* próprio que realiza testes de credenciais com *parsing* dinâmico de respostas HTTP para propagação de *tokens* CSRF; difere da abordagem de

[Sacramento et al. 2024] por integrar a propagação de *tokens* a um módulo autônomo de teste de credenciais, em vez de aos *scripts* de autenticação do próprio ZAP. O módulo é etapa de coleta e não constitui contribuição deste artigo.

5.3. Coleta, Análise e Simulação

Ao final de cada execução, os relatórios JSON são submetidos a um *script Python* [2] que normaliza achados heterogêneos em uma base unificada, em que cada achado é caracterizado por ferramenta de origem, camada (`APP` ou `INFRA`), CWE, severidade CVSS quando aplicável e localização. A função central confronta cada achado com a matriz de vulnerabilidades conhecidas e classifica os pares correspondentes conforme a Seção 4.1. Quatro vulnerabilidades fora do escopo de automação são documentadas mas não participam da avaliação das políticas, pois exigem *payloads* manuais, interação humana ou compreensão de lógica de negócio: *File Inclusion* (CWE-98), *File Upload* (CWE-434), *Insecure CAPTCHA* (CWE-804) e *Authorisation Bypass* (CWE-639).

A matriz de referência é declarada como estrutura de dados versionada no próprio *script*: cada vulnerabilidade é associada ao seu CWE, à camada em que se manifesta (`APP` ou `INFRA`) e, quando aplicável, ao critério de correspondência agregada (para achados de infraestrutura como *Outdated OS*) ou à marcação de exclusão do escopo de automação (para casos que exigem interação manual, como *File Upload*). A matriz completa, com as 17 vulnerabilidades, acompanha o pacote de artefatos.

A simulação das três políticas (P1, P2 e P3) da Seção 4.3 é realizada pelo mesmo *script*, que processa a base unificada e, para cada política e estágio, computa: número de achados elegíveis, bloqueios disparados e composição por classe de detecção. A simulação é determinística e opera sobre os dados coletados, sem reexecução do *pipeline*. O critério de elegibilidade considera apenas achados associados a vulnerabilidades da matriz DVWA ou achados sem matriz mas dentro do escopo do estágio. O *threshold* de severidade é o CVSS v3 do NVD para CVEs do *Container Scan*, e um *proxy* normalizado para ferramentas que não emitem CVSS, derivado da severidade nativa reportada (por exemplo, achados Semgrep `ERROR` mapeiam para CVSS-*proxy* de 7,5). Para os três achados agregados que representam o estado da imagem, adota-se o CVSS máximo entre as CVEs subjacentes como *proxy* de severidade do agregado.

6. Resultados e Análise

6.1. Caracterização dos Achados

A execução do *pipeline* emitiu 1.750 ocorrências brutas distribuídas entre as seis camadas instrumentadas (Tabela 2). O *Container Scan* representa 90% do volume total, refletindo o estado da imagem base *Debian 9.5*, em *End of Support Life*[3].

Das 1.575 CVEs do *Container Scan*, 1.563 (99,2%) possuem pontuação CVSS v3 atribuída pelo NVD, com média 6,98 e distribuição de 17,3% *Critical* (CVSS $\geq$ 9,0),

[2]Disponível no pacote de artefatos: `https://github.com/Machada1/sbseg2026-deteccao-direta-indireta`

[3]O ZAP *Active Scan* reporta 14 *plugins* distintos (661 instâncias agregadas, dominadas pelo *User Agent Fuzzer*); a Tabela 2 reflete os *plugins* distintos.

35,9% *High*, 43,4% *Medium* e 2,7% *Low*. As 89 categorias CWE distintas identificadas refletem o perfil esperado de pacotes legados em código nativo C/C++, com predominância de fraquezas de manipulação de memória: CWE-125, CWE-787, CWE-190 e CWE-476 respondem por mais de 40% das ocorrências.

Tabela 2. Volume bruto de achados por ferramenta (ZAP *Active Scan*: *plugins* distintos disparados).

Ferramenta	Tipo de teste	Achados
Trivy Container	Container Scan	1.575
Semgrep	SAST	77
Checkov	IaC Scan	63
OWASP ZAP (Baseline)	DAST passivo	19
OWASP ZAP (Active)	DAST ativo (plugins distintos)	14
Brute Force	Teste customizado	2
Trivy FS	SCA	0
Total		**1.750**

6.2. Granularidade Mista e Conjunto de Avaliação

Aplicar os 1.750 achados brutos diretamente na simulação distorceria a interpretação, pois cada CVE do *Container Scan* seria uma decisão individual de bloqueio, o que não corresponde à prática operacional, em que o estado da imagem é uma única decisão. Aplicamos, portanto, granularidade mista: SAST, DAST, IaC, SCA e *Brute Force* mantêm granularidade fina (cada achado é uma decisão); o *Trivy Container* é decomposto em três agregados sintéticos (um para *Outdated OS*, representando o estado EOSL; um para *Outdated Packages*, agrupando CVEs com `FixedVersion`; e um para as demais CVEs sem correspondência na matriz), somados a 48 achados individuais cujo CWE coincide com vulnerabilidade aplicacional da matriz. O conjunto resultante tem 226 achados (Tabela 3).

Tabela 3. Composição do conjunto de avaliação (226 achados) após a granularidade mista.

Origem	Tipo	Qtd.
Semgrep	individual	77
Checkov	individual	63
OWASP ZAP (Baseline)	individual	19
OWASP ZAP (Active)	individual	14
Brute Force	individual	2
Trivy Container (CWE coincidente APP)	individual	48
Trivy Container (*Outdated OS*)	agregado	1
Trivy Container (*Outdated Packages*)	agregado	1
Trivy Container (demais CVEs do Sistema Operacional)	agregado	1
Total		**226**

A linha mais relevante é a dos 48 achados do *Container Scan* cujo CWE coincide com o de vulnerabilidade aplicacional da matriz: cada um é uma CVE real em pacote do sistema operacional cuja classificação CWE sugere correspondência a uma vulnerabilidade implementada no código, correspondência que, conforme discutido na Seção 4,

é apenas indireta. As demais 1.527 CVEs representam fraquezas da *stack* ortogonais à matriz DVWA: riscos legítimos da imagem, mas independentes da matriz adotada.

6.3. Cobertura DVWA e Classificação

A Tabela 4 apresenta a matriz de cobertura das 13 vulnerabilidades automatizáveis. Considera-se o *status direta* quando há pelo menos um achado direto associado à vulnerabilidade, independentemente da existência adicional de achados indiretos; *apenas indireta* quando todos os achados associados são indiretos; e *NÃO detectada* na ausência de qualquer achado correspondente. O *pipeline* alcança cobertura total (13/13), mas com distribuição assimétrica: 10 vulnerabilidades são detectadas diretamente por pelo menos uma ferramenta cuja camada coincide com a da vulnerabilidade. As três restantes (CSRF, *Weak Session IDs* e *Open HTTP Redirect*) são detectadas apenas indiretamente, por CVEs do *Container Scan* cujo CWE coincide com o da vulnerabilidade implementada.

Tabela 4. Cobertura das 13 vulnerabilidades automatizáveis da DVWA, com contagem de achados diretos (#dir.) e indiretos (#ind.).

Vulnerabilidade	CWE	Camada	#dir.	#ind.	Status
SQL Injection	CWE-89	APP	15	8	direta
Cross-Site Scripting (XSS)	CWE-79	APP	7	15	direta
Command Injection	CWE-78	APP	11	2	direta
CSRF	CWE-352	APP	0	1	apenas indireta
Weak Session IDs	CWE-330	APP	0	2	apenas indireta
Brute Force	CWE-307	APP	1	0	direta
Open HTTP Redirect	CWE-601	APP	0	8	apenas indireta
JavaScript Attacks	CWE-749	APP	4	0	direta
Content Security Policy Bypass	CWE-693	APP	6	0	direta
Default Credentials	CWE-798	APP	1	0	direta
Exposed MySQL	CWE-284	APP	11	12	direta
Outdated OS	CWE-1104	INFRA	1	0	direta
Outdated Packages	CWE-1104	INFRA	1	0	direta

A assimetria observada é consistente com desafios documentados na literatura sobre DAST em CI/CD, em particular o *overhead* de exercitar fluxos autenticados complexos com configurações de *scan* típicas de pipelines automatizados [Rangnau et al. 2020]. A inspeção do *Active Scan* confirma esse padrão: dos 14 *plugins* disparados, apenas o detector de *SQL Injection* (`pluginId` 40018) executou ataque ativo sobre parâmetro de aplicação; os demais foram verificações passivas de cabeçalhos e varreduras de *User Agent*, que não cobrem CSRF, fraqueza de sessão ou redirecionamento aberto.

A consequência é que três vulnerabilidades aplicacionais ficam sinalizadas exclusivamente por evidência indireta, vinda do *Container Scan*. Sem a distinção direto/indireto, o relatório de cobertura indicaria 13/13 detectadas; com a distinção, fica explícito que 3 das 13 dependem de evidência cuja relação semântica com a aplicação é apenas presumida. Considerando os 226 achados elegíveis à simulação de políticas, 58 (25,7%) são diretos em relação a alguma vulnerabilidade DVWA, 48 (21,2%) são indiretos e 120 (53,1%) não correspondem a nenhuma entrada da matriz.

6.4. Simulação Comparativa das Políticas

A Tabela 5 apresenta o resultado da simulação das três políticas sobre os 226 achados.

Tabela 5. Bloqueios por política sobre os 226 achados, por classe de detecção. P1: CVSS $\geq 9{,}0$ uniforme; P2: CVSS $\geq 7{,}0$ uniforme; P3: CVSS $\geq 7{,}0$ com indiretos não bloqueantes.

Política	Bloqueios	% bloq.	Diretos	Indiretos	Sem matriz
P1	5	2,2%	2	2	1
P2	80	35,4%	29	24	27
P3	56	24,8%	29	0	27

O efeito quantitativo do tratamento diferenciado é direto: P3 produz 24 bloqueios a menos que P2 (redução de 30% em volume bloqueante), e essa diferença é exatamente o número de achados indiretos que P2 bloqueia e P3 não. Composicionalmente, 30% (24/80) dos bloqueios de P2 são detecções indiretas; P3 elimina essa fração e reserva o bloqueio para achados cuja semântica corresponde a falha diretamente observada. A distribuição por estágio confirma a concentração da diferença: P1 dispara todos os 5 bloqueios no *Build Gate*; P2 dispara 51 no *Commit*, 27 no *Build* e 2 no *Deploy*; P3 mantém 51 no *Commit* e 2 no *Deploy*, mas reduz o *Build Gate* de 27 para 3, todos diretos. Os 51 bloqueios no *Commit Gate* de P2 e P3 correspondem a achados Semgrep classificados como `ERROR`, mapeados a CVSS-*proxy* de 7,5 na normalização.

Em termos de cobertura bloqueante (vulnerabilidades DVWA com ao menos um achado disparando bloqueio), P1 cobre 3 de 13, P2 cobre 11 de 13 e P3 cobre 8 de 13; a cobertura sinalizada pelo *pipeline*, independente de bloqueio, mantém-se em 13 de 13 em todas as políticas. As três vulnerabilidades em que P2 supera P3 são CSRF, *Weak Session IDs* e *Open HTTP Redirect*, cuja detecção pelo *pipeline* é apenas indireta (Tabela 4).

A redução da cobertura bloqueante em P3 não constitui virtude do modelo, mas *trade-off* explícito. P3 sacrifica cobertura bloqueante sempre que a cadeia de detecções diretas tem lacunas, e essas lacunas, no experimento, recaem sobre as três vulnerabilidades discutidas na Seção 6.3, cuja detecção direta exigiria *plugins* ativos do DAST atuando sobre formulário autenticado com *token* CSRF, sobre análise estatística de identificadores de sessão e sobre indução controlada de redirecionamento. Sob P2, o *pipeline* bloqueia o *build* apoiado em CVEs do *Container Scan* cujo CWE coincide com essas três vulnerabilidades, ou seja, sobre evidência taxonômica que não confirma exploração efetiva. Sob P3, esses casos são registrados como informação no *Build Gate*, sem disparar bloqueio, e a ausência de detecção direta correspondente no *Deploy Gate* torna-se observável ao operador. A escolha entre P2 e P3 é, em última análise, uma decisão sobre qual erro a equipe prefere: bloquear apoiado em coincidência taxonômica (P2) ou sinalizar sem bloquear quando a cadeia de detecção direta apresenta lacunas (P3).

6.5. Ameaças à Validade

Validade externa. O experimento foi conduzido sobre uma única aplicação-alvo, a DVWA, configurada em nível de segurança baixo, em um único provedor de nuvem. A DVWA é uma aplicação intencionalmente vulnerável, com volumetria de falhas e densidade de padrões inseguros incomuns em sistemas de produção, e a configuração em nível baixo maximiza a detectabilidade. A taxa de detecção e a distribuição direto/indireto observadas tendem a diferir em aplicações reais, em que se espera volume reduzido de SAST acompanhado do surgimento de falsos positivos e deslocamento do eixo de risco conforme a arquitetura. A escolha pela DVWA decorre da disponibilidade de uma ma-

triz de vulnerabilidades conhecidas *a priori*, que em aplicações reais teria que ser derivada de modelagem de ameaças ou inventário de incidentes, com incerteza adicional. Essa derivação desloca, e não elimina, a incerteza: a classificação permanece bem definida dada uma matriz, mas sua utilidade passa a depender da qualidade e da manutenção desta. Mitigam o risco de má classificação o fato de achados indiretos serem registrados, não descartados, e de achados sem correspondência permanecerem sujeitos ao critério de severidade do estágio. Generalizar as proporções exige replicação em múltiplos alvos.

Validade interna. Duas escolhas metodológicas merecem ressalva. Primeiro, o conjunto de avaliação resulta da granularidade mista, em que três achados agregados representam o estado da imagem base. A escolha do CVSS máximo entre as CVEs subjacentes como *proxy* de severidade do agregado favorece o disparo de bloqueios sob políticas estritas e pode superestimar a ação bloqueante atribuída a esses agregados; alternativas como CVSS médio ou mediana levariam a contagens distintas. Segundo, o *Active Scan* do OWASP ZAP foi executado com a política de *scan* padrão de integrações CI/CD, que prioriza tempo de execução sobre exaustividade. *Plugins* ativos que dependem de fluxos autenticados complexos (avaliação de *tokens* CSRF, análise de aleatoriedade de identificadores de sessão e indução de redirecionamento aberto) não foram exercitados de forma sistemática. Essa configuração reflete uso pragmático em produção, alinhada aos desafios de integração de DAST em *pipelines* CI/CD discutidos na literatura [Rangnau et al. 2020, Rajapakse et al. 2022], mas limita a cobertura direta do *Deploy Gate* no experimento e é o principal fator que explica a dependência de detecções indiretas para CSRF, *Weak Session IDs* e *Open HTTP Redirect*.

Validade de construto. A classificação direto/indireto apoia-se parcialmente em coincidência taxonômica de CWE, com a ressalva de que tal coincidência entre ferramentas em camadas distintas não implica equivalência semântica. O modelo trata isso explicitamente, distinguindo as classes na decisão de bloqueio, mas a definição de quais ferramentas constituem detecção direta exige conhecimento prévio da matriz e do *pipeline*, declarado manualmente neste experimento. Adicionalmente, 12 das 1.575 CVEs sem pontuação CVSS pelo NVD foram tratadas como severidade neutra, o que pode subestimar bloqueios sobre falhas reais com pontuação ainda em atribuição.

7. Conclusão e Trabalhos Futuros

Este trabalho propôs e avaliou empiricamente a distinção entre detecções diretas e indiretas em *pipelines* DevSecOps multicamada e demonstrou seu uso na calibração de portões de segurança progressivos, por meio de um modelo de classificação, portões em quatro estágios e três políticas simuladas sobre os achados de um *pipeline* de seis camadas.

A avaliação contra a DVWA mostrou que 21,2% dos achados elegíveis à decisão de portão são detecções indiretas, e que a política proposta (P3), ao tratá-los como registro informativo em vez de bloqueante, reduz em 30% o volume de bloqueios em relação à política estrita uniforme. Essa redução vem com um *trade-off* explícito: P3 perde cobertura bloqueante sobre vulnerabilidades cuja única evidência no *pipeline* é indireta, situação que recai sobre três vulnerabilidades cuja detecção direta dependeria de *plugins* de DAST que a configuração de *scan* não exercitou (Seção 6.5). A cobertura sinalizada pelo *pipeline* permanece em 13 de 13 vulnerabilidades em todas as políticas; o que muda é o estágio e a forma como cada uma é tratada na decisão automatizada. O pacote de

artefatos com os relatórios JSON brutos, o *script* de normalização e a matriz de referência está disponível publicamente para reprodução do experimento[4].

Como trabalho futuro, três direções se destacam: replicação em múltiplos alvos com matrizes de referência distintas, incluindo aplicações de produção em que a matriz é derivada de modelagem de ameaças; integração do estágio *Runtime Watch* com telemetria efetiva, fechando o ciclo de realimentação previsto pelo modelo; e integração da classificação direto/indireto a plataformas ASPM, avaliando se a inclusão desse eixo melhora a tomada de decisão em cenários de alta volumetria.

Referências

Ahmed, Z. and Francis, S. C. (2019). Integrating security with DevSecOps: Techniques and challenges. In *2019 Int. Conf. on Digitization (ICD)*, pages 178–182.

Bernardino, N. A., Sequeira, B., Piza, E., Henriques, F., Neves, F., and Reis, C. I. (2024). Enhancing DevSecOps: Three custom tools for continuous security. In *2024 IEEE 11th Int. Conf. on Cyber Security and Cloud Computing (CSCloud)*, pages 53–58.

Chen, T. and Suo, H. (2022). Design and practice of security architecture via DevSecOps technology. In *2022 IEEE 13th Int. Conf. on Software Engineering and Service Science (ICSESS)*, pages 310–313.

Dewhurst, R. (2026). Damn vulnerable web application (DVWA). `https://github.com/digininja/DVWA`. Acessado em: 09 fev. 2026.

Díaz, E. O., Muñoz, M., and Mejía, J. (2019). Responsive infrastructure with cybersecurity for automated high availability DevSecOps processes. In *2019 8th Int. Conf. On Software Process Improvement (CIMPS)*, pages 1–9.

Efendi, M., Raharjo, T., and Suhanto, A. (2021). DevSecOps approach in software development case study: Public company logistic agency. In *2021 Int. Conf. on Informatics, Multimedia, Cyber and Information System (ICIMCIS)*, pages 96–101.

Feio, C., Santos, N., Escravana, N., and Pacheco, B. (2024). An empirical study of DevSecOps focused on continuous security testing. In *2024 IEEE European Symp. on Security and Privacy Workshops (EuroS&PW)*, pages 610–617.

FIRST (2026). Common vulnerability scoring system (CVSS). `https://www.first.org/cvss/`. Acessado em: 26 mar. 2026.

Forsgren, N., Humble, J., and Kim, G. (2018). *Accelerate: The Science of Lean Software and DevOps*. IT Revolution, Portland.

Humble, J. and Farley, D. (2010). *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley Professional, Boston.

Kujavski, L. M., Penteado, U., Almeida, P. L. d., and Grégio, A. (2024). Obsolescência não-programada: Análise do uso de *Software* desatualizado em ambiente de produção. In *Anais do XXIV SBSeg*, pages 508–521. SBC.

[4] `https://github.com/Machada1/sbseg2026-deteccao-direta-indireta`.

Kushwaha, M. K., David, P., and Suseela, G. (2024). Automation and DevSecOps in financial systems. In *2024 IEEE Int. Conf. on Electronics, Computing and Communication Technologies (CONECCT)*, pages 1–6.

Marandi, M., Bertia, A., and Silas, S. (2023). Implementing and automating security scanning to a DevSecOps CI/CD pipeline. In *2023 World Conf. on Communication & Computing (WCONF)*, pages 1–6.

Masood, A. and Java, J. (2015). Static analysis for web service security. In *2015 IEEE Int. Symp. on Technologies for Homeland Security (HST)*, pages 1–6.

MITRE (2026a). Common vulnerabilities and exposures (CVE). `https://cve.mitre.org/`. Acessado em: 26 mar. 2026.

MITRE (2026b). Common weakness enumeration (CWE). `https://cwe.mitre.org/`. Acessado em: 26 mar. 2026.

NIST (2026). National vulnerability database (NVD). `https://nvd.nist.gov/`. Acessado em: 26 mar. 2026.

Open Worldwide Application Security Project (2025). OWASP Top Ten: 2025. `https://owasp.org/Top10/2025/`. Acessado em: 26 mar. 2026.

Ponce, L. M., Ribeiro, I., Oliveira, E., Cunha, , Hoepers, C., Steding-Jessen, K., Chaves, M. H. P. C., Guedes, D., and Meira Jr., W. (2024). Identificação de serviços e dispositivos em dados de motores de busca para o enriquecimento de análise de vulnerabilidades. In *Anais do XXIV SBSeg*, pages 367–382. SBC.

Putra, A. M. and Kabetta, H. (2022). Implementation of DevSecOps by integrating static and dynamic security testing in CI/CD pipelines. In *2022 IEEE Int. Conf. of Computer Science and Information Technology (ICOSNIKOM)*, pages 1–6.

Rajapakse, R. N., Zahedi, M., Babar, M. A., and Shen, H. (2022). Challenges and solutions when adopting DevSecOps: A systematic review. *Information and Software Technology*, 141:106700.

Rangnau, T., v. Buijtenen, R., Fransen, F., and Turkmen, F. (2020). Continuous security testing: A case study on integrating dynamic security testing tools in CI/CD pipelines. In *2020 IEEE 24th Int. EDOC Conf.*, pages 145–154.

Ribeiro, D. S., Lemos, R., da Ponte, F. R. P., Mattos, C. L. C., and Rodrigues, E. B. (2024). Classificação de risco de vulnerabilidades de segurança via processos gaussianos e aprendizado ativo. In *Anais do XXIV SBSeg*, pages 107–122. SBC.

Sacramento, L. S. C., Cunha, , Cardoso, G. P. d. O., Souza, A., Franco, A., and Oliveira, L. B. (2024). Automação de autenticação para testes de segurança em aplicações *Web* no ZAP. In *Anais do XXIV SBSeg*, pages 760–766. SBC.

Sousa, J. O. d., Farias, B. C. d., Lima Filho, E. B. d., and Cordeiro, L. C. (2024). Trust, but verify: Developer behavior in mitigating vulnerabilities in open-source projects. In *Anais do XXIV SBSeg*, pages 616–631. SBC.

Sun, X., Cheng, Y., Qu, X., and Li, H. (2021). Design and implementation of security test pipeline based on DevSecOps. In *2021 IEEE 4th Int. Conf. on Advanced Information Management (IMCEC)*, volume 4, pages 532–535.