SecDevAgent: Agentes de segurança para validações de Falsos positivos de scans SAST

Thiago Lotufo[1], **Vitor Maurício Rodrigues**[1], **Eduardo Quintanilha**[1]

[1]Globo Comunicação e Participações S/A
700 BL2, Av. das Américas, 1° Andar, Rio de Janeiro, 22640-100, RJ – Brasil

thiago.lotufo@g.globo, vitor.rodrigues@g.globo, eduardo.quintanilha@g.globo

Abstract. *Static Application Security Analysis (SAST) tools are widely used in DevSecOps pipelines to detect vulnerabilities in the early stages of software development, ensuring Shift-Left Security. However, one of the biggest problems with this technology is the false positives generated, consuming engineering teams' time and reducing developers' confidence in security tools. This article presents SecDevAgent, an artificial intelligence agent built on LangChain and large-scale language models (LLMs) via Google Vertex AI, designed to automatically triage SAST findings and classify them as confirmed vulnerabilities, false positives, or inconclusive. The agent clones the target repository, inspects the reported file, discovers related files through symbol-based heuristics and code search, and determines whether a vulnerability is a false positive or not. This article describes the agent's architecture, its tool-oriented reasoning cycle, integration modes (CLI and HTTP API with support for asynchronous tasks via MongoDB), and presents preliminary results compared to manual vulnerability screenings and SAST scan results.*

Resumo. *Ferramentas de Análise Estática de Segurança de Aplicações (SAST) são amplamente utilizadas em pipelines de DevSecOps para detectar vulnerabilidades nas fases iniciais do desenvolvimento de software, garantindo o Shift-Left Security. Entretanto, um dos maiores problemas desta tecnologia, são os falsos positivos gerados, consumindo o tempo das equipes de engenharia e reduzindo a confiança dos desenvolvedores nas ferramentas de segurança. Este artigo apresenta o SecDevAgent, um agente de inteligência artificial construído sobre LangChain e modelos de linguagem de grande escala (LLMs) via Google Vertex AI, projetado para triar automaticamente achados de SAST e classificá-los como vulnerabilidade confirmada, falsos positivos ou inconclusivos. O agente clona o repositório alvo, inspeciona o arquivo reportado, descobre arquivos relacionados por heurísticas baseadas em símbolos e busca de código, e determina se uma vulnerabilidade é um falso positivo ou não. Neste artigo, está descrita a arquitetura do agente, seu ciclo de raciocínio orientado a ferramentas, os modos de integração (CLI e API HTTP com suporte a tarefas assíncronas via MongoDB) e apresentamos resultados preliminares, em comparação com triagens manuais de vulnerabilidades e resultados do scan SAST.*

1. Introdução

A cultura *DevSecOps* reúne práticas, processos e ferramentas que garantem o Shift-Left Security [CrowdStrike 2024]: ferramentas de análise estática de código (SAST, *Static Application Security Testing*) buscam vulnerabilidades antes que estas cheguem à produção.

Tais achados, no entanto, precisam de validação para descartar falsos positivos e estimar o impacto real da vulnerabilidade.

Ferramentas SAST como Semgrep, Bandit e Gosec são amplamente adotadas por possuírem versões Open Source gratuitas. Como analisam apenas padrões de escrita, sem contexto geral da aplicação, geram taxas de falsos positivos entre 30% e 80%, dependendo da ferramenta e do projeto [Johnson et al. 2013], causando **fadiga de alertas**: falsos positivos ofuscam vulnerabilidades com risco real de exploração.

A validação manual desses achados exige profissionais com conhecimento técnico avançado. Modelos de linguagem de grande escala (LLMs) têm demonstrado capacidade crescente de compreensão e raciocínio sobre código-fonte [IBM 2024], permitindo, ao receberem o contexto adequado, validar o potencial de exploração de uma vulnerabilidade e seu impacto para o produto e o usuário.

Este artigo tem como objetivo apresentar o **SecDevAgent**, um agente de IA com a capacidade de realizar a triagem dos resultados de um scan SAST, confirmando se os achados são, de fato, uma vulnerabilidade de segurança. O agente é construído sobre LangChain [Chase 2022] e modelos Gemini via Google Vertex AI, e implementa o padrão ReAct (*Reasoning + Acting*) [Yao et al. 2023] para raciocinar iterativamente sobre o código do repositório. As principais contribuições deste trabalho são:

- Arquitetura de agente orientada a ferramentas para análise contextual de vulnerabilidades em repositórios reais;
- Protocolo de descoberta de arquivos relacionados por extração de símbolos combinada com busca por expressão regular;
- Integração via CLI e API REST, com execução assíncrona e persistência em MongoDB;
- Avaliação preliminar sobre achados reais de SAST em projetos da Globo.

2. Trabalhos Relacionados

Ferramentas SAST como Semgrep [Semgrep, Inc. 2020], Bandit [PyCQA 2014] e Gosec [securego 2016] implementam correspondência de padrões e análise de fluxo de dados. Orquestradores como o HuskyCI [Globo 2019] integram múltiplas dessas ferramentas em pipelines CI/CD, identificando linguagens e executando ferramentas Open Source em busca de vulnerabilidades. Nenhuma dessas abordagens, porém, trata o problema dos falsos positivos gerados.

Trabalhos anteriores exploraram LLMs para detecção de vulnerabilidades de software [Chen et al. 2025], com análise profunda de código e modelos para potencializar essa descoberta. A demanda por modelos focados em segurança da informação é crescente, com grandes players de IA desenvolvendo produtos cada vez mais robustos [Anthropic 2025]. O padrão ReAct [Yao et al. 2023] demonstrou que LLMs podem raciocinar iterativamente, intercalando reflexão com chamadas a ferramentas externas. Este trabalho aplica esse paradigma à triagem de segurança, equipando o LLM com ferramentas especializadas de navegação e compreensão de repositórios, o que o diferencia de abordagens que usam LLMs de forma estática para classificação direta de achados.

3. SecDevAgent

Nesta seção, apresenta-se o SecDevAgent, um agente de IA para triagem automática de falsos positivos em achados de ferramentas SAST. Serão descritas sua entrada de dados, arquitetura, o ciclo de raciocínio do agente, as ferramentas disponíveis e o formato do veredicto produzido.

3.1. Entrada de Dados

O SecDevAgent recebe um objeto JSON padronizado descrevendo o achado SAST: identificador (`id`), título (`title`), descrição (`description`), severidade (`severity`), caminho do arquivo (`file_path`), linha opcional (`line`), regra do scan (`rule_id`) e evidência de código (`evidence`). Também são informados a localização do repositório (URL Git ou caminho local) e o modelo LLM a ser utilizado.

3.2. Arquitetura

O sistema é composto por quatro camadas principais: *(i)* interface de entrada (CLI ou API HTTP); *(ii)* módulo de preparação do repositório; *(iii)* núcleo do agente baseado no padrão ReAct; e *(iv)* conjunto de ferramentas de análise de repositório.

A Figura 1 ilustra a arquitetura geral do SecDevAgent.

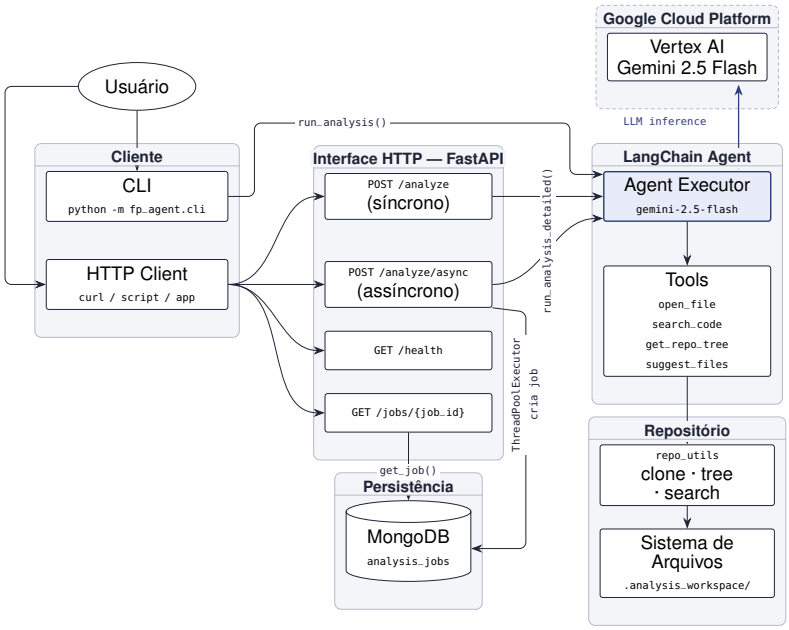

Figura 1. Arquitetura geral do SecDevAgent.

Na preparação, o módulo `repo_utils` clona o repositório remoto (`git clone`) ou copia um repositório local para um diretório de trabalho isolado, garantindo que o agente opere sobre uma cópia imutável do código, imune a alterações posteriores do usuário. O núcleo do agente é construído sobre a abstração `create_agent` do LangChain, com um *system prompt* que instrui o LLM a atuar como engenheiro sênior de AppSec e `temperature=0` para maximizar determinismo e reprodutibilidade.

3.3. Ciclo de Raciocínio (ReAct)

O ciclo de execução segue o padrão ReAct [Yao et al. 2023]: *(i)* o LLM analisa o contexto e decide qual ferramenta invocar; *(ii)* a ferramenta é executada e seu resultado retorna ao LLM; *(iii)* o LLM incorpora o resultado ao raciocínio e decide a próxima ação; *(iv)* o ciclo se repete até a resposta final.

O *prompt* instrui o agente a seguir uma ordem de investigação: *(i)* inspecionar a árvore do repositório; *(ii)* abrir o arquivo reportado; *(iii)* identificar arquivos relacionados; *(iv)* abrir ao menos dois arquivos adicionais relevantes; e *(v)* realizar buscas textuais complementares se necessário.

Nos testes, o agente utilizou o Gemini 2.5 Flash [Google DeepMind 2025], cuja janela de contexto se aproxima de 1 milhão de tokens. Em repositórios com arquivos muito grandes, pode ser necessário reduzir o contexto enviado ao modelo.

3.4. Ferramentas de Repositório

O agente dispõe de quatro ferramentas especializadas para navegação e análise do repositório:

- **`get_repo_tree`**: retorna representação compacta em árvore dos arquivos (até 1.200 entradas), ignorando diretórios irrelevantes (`node_modules`, `.git`, `__pycache__`, testes e documentação), para o LLM compreender a estrutura do projeto antes de investigar;
- **`open_file`**: abre um arquivo por caminho relativo, retornando até 12.000 caracteres, com proteção contra *path traversal* [OWASP Foundation 2024];
- **`suggest_related_files`**: extrai por expressões regulares candidatos a símbolos do arquivo analisado (classes, funções, variáveis, importações) e busca via ripgrep suas definições e pontos de chamada em outros arquivos, expandindo o contexto de uso do código vulnerável;
- **`search_code`**: busca por expressão regular no repositório via ripgrep, retornando até 30 resultados com caminho e número de linha.

3.5. Formato de Saída do Agente

Ao concluir a investigação, o agente produz uma resposta estruturada com os seguintes campos:

- **VERDICT**: `false_positive`, `true_positive` ou `inconclusive`;
- **CONFIDENCE**: `low`, `medium` ou `high`;
- **REASONING**: explicação concisa fundamentada nos arquivos analisados;
- **FILES_ANALYZED**: lista dos arquivos inspecionados;
- **FOLLOW_UP_ACTIONS**: ações recomendadas para casos inconclusivos.

4. Modos de Integração

O SecDevAgent oferece três formas de integração: CLI, API HTTP síncrona e API HTTP assíncrona com persistência em MongoDB.

A **CLI** permite uso direto por engenheiros de segurança e integração em scripts de automação, recebendo a URL do repositório, o JSON da vulnerabilidade, o modelo LLM e uma flag de rastreamento (`-trace`) que exibe o ciclo de raciocínio completo.

A **API HTTP síncrona**, implementada com FastAPI, expõe o endpoint `POST /analyze`, sem suporte a *streaming*: a requisição aguarda a conclusão da análise antes de retornar o relatório e os eventos de rastreamento.

A **API HTTP assíncrona** expõe `POST /analyze/async`, que enfileira a análise e retorna imediatamente um `job_id`, processado em segundo plano por um pool de *workers*. O cliente consulta o estado via `GET /jobs/{job_id}` (`queued`, `running`, `succeeded` ou `failed`). O MongoDB persiste o *payload*, status, *timestamps* e resultado completo, permitindo auditoria e evitando revalidar uma vulnerabilidade já analisada. A Figura 2 ilustra esse fluxo.

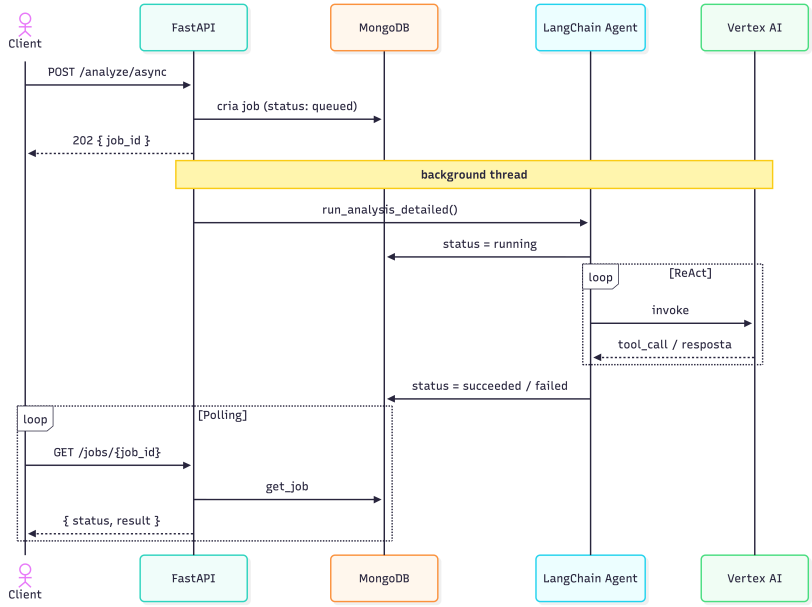

Figura 2. Fluxo de execução no modo assíncrono com persistência em MongoDB.

5. Avaliação Experimental

5.1. Configuração Experimental

Para avaliar o SecDevAgent, selecionamos um conjunto de achados de SAST gerados pelo HuskyCI [Globo 2019] e pelo Gitlab Security Scan [GitLab Inc. 2025], executados em projetos da Globo, hospedados no Gitlab. Os achados foram previamente validados como vulnerabilidades confirmadas e falsos positivos, constituindo o *ground truth* da avaliação. O agente foi configurado com o modelo `gemini-2.5-flash` via Google Vertex AI, com `temperature=0`.

5.2. Análise Quantitativa

Foi realizada uma análise prévia das vulnerabilidades, antes da validação do SecDevAgent. Para o **Repositório A**, de um total de 17 vulnerabilidades, 15 eram vulnerabilidades confirmadas e 2 eram falsos positivos. Já para o **Repositório B**, de um total de 34 vulnerabilidades, todas as 34 eram falsos positivos.

A Tabela 1 apresenta a taxa de acerto do SecDevAgent, quando comparado com validações manuais

Tabela 1. Taxas de acerto do SecDevAgent

Repositório	Taxa de Acerto	Resultados Inconclusivos
Repositório A	88%	0
Repositório B	88%	4

A Tabela 2 consolida os resultados de vulnerabilidades confirmadas e de falsos positivos do SecDevAgent, comparados ao resultado real (validação humana) em cada repositório.

Tabela 2. Resultados do SecDevAgent por repositório avaliado

Repo.	Total	Conf. (Ag.)	Conf. (Hum.)	FP (Ag.)	FP (Hum.)
A	17	16 (94,1%)	15 (88%)	1 (5,9%)	2 (12%)
B	34	0 (0,0%)	0 (0,0%)	30 (88,2%)	34 (100%)
Total	**51**	**16**	**15**	**31**	**36**

5.3. Análise Qualitativa

A análise qualitativa revelou padrões recorrentes. Em achados de algoritmos criptográficos fracos (ex.: MD5), o agente classifica corretamente como falso positivo quando a `suggest_related_files` identifica que o hash é usado para indexação ou verificação de integridade não crítica, e não para proteção de dados sensíveis (senhas, pagamentos) — algo que ferramentas SAST não conseguem discernir, por confirmarem apenas o padrão de uso e não seu contexto.

Já achados de SQL Injection ou Insecure Deserialization tendem a ser confirmados quando a mesma ferramenta identifica dados externos fluindo sem sanitização até o ponto vulnerável. Casos inconclusivos ocorrem principalmente quando o código relevante está distribuído em muitos arquivos com dependências profundas, limitando a análise à janela de contexto do LLM.

Observou-se ainda dificuldade do modelo em compreender, de forma autônoma, convenções específicas de organização de pastas (por exemplo, definição heterogênea de testes unitários entre times), o que impacta os resultados finais.

6. Considerações Finais

Neste artigo, foi apresentado o SecDevAgent, um agente de IA para triagem automática de falsos positivos em achados de ferramentas SAST. O agente combina o raciocínio iterativo de LLMs com ferramentas especializadas de navegação de repositório para produzir veredictos fundamentados em evidências concretas do código, replicando o processo de investigação de um profissional de segurança especializado.

A arquitetura modular baseada em LangChain e Google Vertex AI permite fácil adaptação a diferentes modelos LLM e integração em pipelines existentes, via CLI, API síncrona ou fluxo assíncrono com persistência. Os resultados preliminares indicam que a abordagem é promissora para reduzir o esforço manual de triagem, especialmente em organizações com alto volume de achados SAST.

Como trabalhos futuros, destacamos: melhorias na janela de contexto para arquivos grandes; treinamento e utilização de modelos proprietários; e Skills voltadas às regras de negócio da Globo, permitindo ao Agente avaliar não só falsos positivos mas o risco e impacto real de cada vulnerabilidade; parsers específicos por linguagem (Python AST, Tree-sitter) para extração mais precisa de dependências; mecanismos de *active learning* com feedback de revisores humanos; e integração nativa em pipelines CI/CD via GitLab CI.

Referências

Anthropic (2025). Project Glasswing: Securing critical software with AI. https://www.anthropic.com/glasswing. Acesso em: 25 mai. 2026.

Chase, H. (2022). LangChain: Building applications with LLMs through composability. https://github.com/langchain-ai/langchain.

Chen, Z., Gu, S., Huang, H., Gu, G., and Huang, J. (2025). LLMs in software security: A survey of vulnerability detection techniques and insights. *arXiv preprint arXiv:2502.07049*. v2.

CrowdStrike (2024). What is shift left security? https://www.crowdstrike.com/en-us/cybersecurity-101/cloud-security/shift-left-security/. Acesso em: 25 mai. 2026.

GitLab Inc. (2025). Application security. https://docs.gitlab.com/user/application_security/. Acesso em: 25 mai. 2026.

Globo (2019). HuskyCI: Orchestrating security tests in CI pipelines. https://github.com/globocom/huskyCI.

Google DeepMind (2025). Gemini 2.5 flash. https://ai.google.dev/gemini-api/docs/models/gemini-2.5-flash. Acesso em: 25 mai. 2026.

IBM (2024). Code LLM: How large language models are changing software development. https://www.ibm.com/think/insights/code-llm. Acesso em: 25 mai. 2026.

Johnson, B., Song, Y., Murphy-Hill, E., and Bowdidge, R. (2013). Why don't software developers use static analysis tools to find bugs? In *Proceedings of the 35th International Conference on Software Engineering (ICSE)*, pages 672–681. IEEE.

OWASP Foundation (2024). Path traversal. https://owasp.org/www-community/attacks/Path_Traversal. Acesso em: 25 mai. 2026.

PyCQA (2014). Bandit: A security linter for Python. https://github.com/PyCQA/bandit.

securego (2016). Gosec: Golang security checker. https://github.com/securego/gosec.

Semgrep, Inc. (2020). Semgrep: Static analysis at ludicrous speed. https://semgrep.dev.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., and Cao, Y. (2023). ReAct: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*.