SAST para detecção de vulnerabilidades em workflows do n8n

João Pedro Schmidt Cordeiro Gustavo Zambonin

4 de dezembro de 2025

Resumo

A crescente adoção de plataformas Low-Code/No-Code (LCNC), como o n8n, tem transformado o desenvolvimento de automações empresariais, democratizando a criação de workflows através de interfaces visuais. Entretanto, essa mudança de paradigma introduz uma lacuna crítica de governança: enquanto a plataforma subjacente é protegida por práticas robustas de segurança, os workflows criados pelos usuários não são submetidos ao mesmo rigor de análise de segurança. Este trabalho propõe o desenvolvimento de uma ferramenta de Análise Estática de Segurança de Aplicações (SAST) específica para workflows do n8n, capaz de detectar automaticamente vulnerabilidades como SQL Injection, Command Injection, SSRF, exposição de credenciais, manuseio inseguro de dados e negação de serviço. A metodologia baseia-se em uma estratégia híbrida que combina o Agentic Radar (focado em riscos de agentes de IA) e o Semgrep (configurado com regras customizadas para vulnerabilidades tradicionais). Além disso, propõe-se o uso de inteligência artificial para acelerar a criação de regras de detecção. A análise revelou coberturas complementares de 16,7% e 66,7% respectivamente, demonstrando a viabilidade da análise estática de estruturas JSON declarativas. A solução contribui para elevar o nível de segurança das automações desenvolvidas em organizações brasileiras, auxiliando na conformidade com a LGPD.

Palavras-chave: SAST, n8n, Low-Code/No-Code, Segurança de Aplicações, Análise Estática, Vulnerabilidades, Automação de Workflows, Inteligência Artificial, LGPD.

Sumário

1 Introdução

A crescente adoção de plataformas Low-Code/No-Code (LCNC) representa uma transformação fundamental no desenvolvimento de aplicações e automações empresariais. O n8n, uma proeminente plataforma de automação de workflows de código aberto, exemplifica essa mudança ao capacitar equipes técnicas a conectar APIs, bancos de dados e serviços através de um editor visual intuitivo. Esta democratização do desenvolvimento, embora acelere drasticamente a criação de automações, introduz desafios críticos de segurança que este trabalho busca endereçar.

A mudança de um modelo de codificação imperativa tradicional para um modelo de configuração declarativa, onde a lógica é definida visualmente e armazenada como objetos JSON, introduz um novo paradigma para a segurança de aplicações. Existe uma lacuna crítica de governança: o "código-fonte"da aplicação (arquivo JSON do workflow) não está sujeito ao mesmo rigor de segurança que a plataforma na qual é executado. As equipes de segurança não podem mais depender exclusivamente das garantias de segurança do fornecedor; em vez disso, devem estabelecer seus próprios processos de garantia para os ativos criados na plataforma.

1.1 Motivação e Contexto

O interesse por este tema surge de uma necessidade prática identificada no ambiente de trabalho, onde a plataforma n8n é utilizada diariamente. Com o crescimento do número de workflows e a expansão do uso da ferramenta para diferentes áreas organizacionais, observou-se a necessidade crítica de averiguar a segurança dos workflows desenvolvidos. Particularmente preocupante é o fato de que muitos usuários que criam workflows não possuem conhecimento técnico aprofundado em segurança, o que pode resultar no desenvolvimento não intencional de vulnerabilidades dentro do sistema.

Esta experiência prática evidencia a lacuna de governança identificada na literatura, onde o "desenvolvedor cidadão"assume responsabilidades de desenvolvimento sem o treinamento formal necessário para identificar riscos de segurança. A própria n8n implementa práticas de

segurança robustas em seu código-fonte, incluindo a utilização de Testes de Segurança de Aplicações. Contudo, essa varredura não se estende, nem poderia se estender, aos workflows criados pelos seus usuários. A responsabilidade pela concepção de workflows seguros é explicitamente delegada ao usuário.

O potencial de impacto no contexto brasileiro é particularmente significativo considerando a crescente digitalização de processos empresariais e a adoção de ferramentas de automação no país. A Lei Geral de Proteção de Dados (LGPD) [1] e outras regulamentações de segurança cibernética criam uma demanda específica por ferramentas que possam garantir a conformidade e segurança de automações empresariais. Uma ferramenta SAST especializada para n8n junto com a rápida criação de regras novas, de acordo com a demanda, utilizando inteligência artificial, pode contribuir significativamente para elevar o nível de segurança das automações desenvolvidas por organizações brasileiras, reduzindo riscos de vazamento de dados e ataques cibernéticos que podem resultar em penalidades regulatórias e danos reputacionais.

1.2 Problema Abordado

O principal desafio de segurança no n8n não é uma falha da plataforma em si, mas sim uma falha potencial na implementação do "usuário-desenvolvedor". Os workflows podem conter seis classes críticas de vulnerabilidades: **SQL Injection (SQLi)**, onde entradas controladas pelo usuário são inseridas em consultas SQL sem sanitização adequada; **Command/Code Injection**, com dados não confiáveis usados em comandos executados no servidor; **Secret Sprawl & Chaining**, envolvendo dispersão de segredos em configurações ou logs; **SSRF**, quando entradas do usuário determinam URLs em requisições HTTP sem validação; **Insecure Data Handling**, com falhas na proteção de dados sensíveis; e **Denial of Service (DoS)**, através de workflows suscetíveis a loops infinitos ou operações intensivas.

A natureza declarativa do JSON do n8n, embora abstraia a complexidade do código tradicional, paradoxalmente torna certos tipos de análise estática mais fáceis e precisas. Ele declara explicitamente as relações de fluxo de dados através do array `connections`, permitindo regras de análise de contaminação (taint analysis) com elevado grau de confiança e baixa taxa de falsos positivos.

1.3 Delimitação do Escopo

Este trabalho foca especificamente no desenvolvimento e validação de uma abordagem híbrida para análise estática de segurança de workflows n8n. O escopo inclui análise estática de arquivos JSON de workflows, avaliação detalhada de duas ferramentas do estado da arte (Agentic Radar e Semgrep), detecção das seis classes principais de vulnerabilidades, desenvolvimento de metodologia para criação de regras Semgrep utilizando inteligência artificial, uso de workflows com vulnerabilidades conhecidas como dataset de validação, e análise de viabilidade técnica com métricas de performance.

Estão excluídos do escopo: implementação completa de uma ferramenta SAST de produção, interface gráfica para usuários finais, análise dinâmica ou execução real de workflows, análise de nós customizados da comunidade, correção automática de vulnerabilidades, otimização de performance de workflows, deploy em ambientes de produção corporativos, e análise de vulnerabilidades em nível de infraestrutura da plataforma n8n.

1.4 Contribuições Esperadas

Este trabalho oferece contribuições técnicas, metodológicas e científicas para segurança em plataformas LCNC. Primeiro, apresenta uma **análise comparativa** sistemática de Agentic Radar e Semgrep, documentando coberturas complementares e aplicabilidade ao contexto n8n. Segundo, fornece **validação de viabilidade** através de demonstração empírica da eficácia da análise estática de estruturas JSON declarativas. Terceiro, propõe uma **abordagem híbrida** que combina ferramentas especializadas e genéricas configuráveis para maximizar cobertura. Quarto, desenvolve **metodologia com IA** utilizando inteligência artificial para acelerar a criação de regras de detecção. Quinto, inclui **documentação de lacunas** identificando limitações e oportunidades para pesquisas futuras.

A relevância para a formação profissional está diretamente alinhada com as tendências emergentes do mercado de tecnologia. O desenvolvimento de competências em análise de segurança para plataformas LCNC representa uma especialização altamente demandada no mercado, posicionando o profissional na interseção entre desenvolvimento de baixo código e segurança cibernética. A capacidade de identificar e mitigar riscos em ambientes de desenvolvimento democratizado torna-se um diferencial competitivo crucial à medida que mais organizações adotam estratégias de desenvolvimento cidadão.

2 Fundamentação Teórica e Estado da Arte

Esta seção apresenta a revisão da literatura relevante e análise crítica das ferramentas existentes para análise estática de segurança em plataformas Low-Code/No-Code, com foco específico na plataforma n8n. A análise concentra-se em duas ferramentas principais que representam abordagens distintas ao problema: o Agentic Radar, uma solução moderna baseada em agentes de IA, e o Semgrep, um motor de análise estática clássico e extensível.

2.1 Técnicas de SAST para Plataformas Low-Code/No-Code

O cenário de ferramentas para análise de segurança estática de workflows n8n é emergente e fragmentado, apresentando abordagens distintas que podem ser categorizadas em ferramentas dedicadas e adaptáveis. É crucial ressaltar que, até o momento, **não existe uma abordagem oficial ou solução consolidada para detecção de vulnerabilidades especificamente em workflows do n8n**. Esta lacuna representa tanto um desafio quanto uma oportunidade significativa para pesquisa e desenvolvimento na área.

2.1.1 Abordagem Moderna: Agentic Radar e IA Agêntica

O Agentic Radar, desenvolvido pela SPLX AI, emerge como a ferramenta de código aberto mais proeminente especificamente projetada para análise de segurança de workflows em plataformas de automação, incluindo o n8n [13]. A ferramenta representa uma abordagem moderna ao problema, incorporando capacidades de Inteligência Artificial Agêntica (Agentic AI) para análise de fluxos de trabalho onde agentes de IA interagem com ferramentas, APIs e serviços externos.

A utilização de IA no Agentic Radar manifesta-se em múltiplas dimensões. Primeiro, a ferramenta implementa análise estática consciente do contexto, capaz de interpretar a estrutura JSON de workflows e construir representações gráficas dos fluxos de dados. Segundo, o modo de teste (`test`) utiliza modelos de linguagem (especificamente a API da OpenAI) para executar

testes adversariais em tempo de execução, simulando ataques de injeção de prompt, tentativas de extração de informações sensíveis e geração de conteúdo prejudicial [14]. Esta capacidade de testar dinamicamente a robustez de sistemas de agentes de IA através de entradas adversariais geradas por outros modelos de linguagem representa uma aplicação sofisticada de IA para segurança.

Terceiro, a ferramenta correlaciona padrões identificados nos workflows com categorias de risco estabelecidas pelo OWASP LLM Top 10, sugerindo a utilização de classificação assistida por IA para mapeamento de vulnerabilidades. Quarto, a geração de relatórios com visualizações gráficas interativas e recomendações de remediação contextualizadas indica o uso de técnicas de geração de linguagem natural para produzir explicações técnicas compreensíveis.

O foco atual do Agentic Radar concentra-se em riscos emergentes específicos de sistemas de agentes de IA: injeção de prompt, vazamento de PII, geração de conteúdo nocivo e disseminação de desinformação. Esta especialização não cobre vulnerabilidades tradicionais de aplicações web, limitando sua cobertura a aproximadamente 16,7% das classes identificadas.

2.1.2 Abordagem Clássica: Semgrep e Análise Baseada em Padrões

O Semgrep, desenvolvido pela Semgrep Inc., representa uma abordagem clássica de análise estática de código, fundamentada em correspondência de padrões consciente de sintaxe [12]. Diferentemente de ferramentas baseadas em expressões regulares, o Semgrep realiza parsing do código-fonte em árvores sintáticas abstratas (AST), oferecendo compreensão semântica da estrutura do código. A ferramenta suporta mais de 30 linguagens de programação, incluindo análise nativa de formatos estruturados como JSON e YAML, tornando-a particularmente relevante para a análise de workflows do n8n.

A arquitetura do Semgrep fundamenta-se em uma linguagem de regras declarativa, onde padrões de detecção são especificados em arquivos YAML. Esta abordagem permite que equipes de segurança desenvolvam conjuntos de regras personalizados sem a complexidade de manipulação direta de ASTs ou construção de compiladores [11]. No contexto do n8n, regras podem ser desenvolvidas para detectar padrões inseguros específicos: concatenação de entradas não confiáveis em consultas SQL, construção dinâmica de URLs em nós HTTP Request baseada em dados externos, uso de expressões dinâmicas em comandos de execução e exposição de credenciais em configurações.

A capacidade de análise de fluxo de dados (dataflow analysis) do Semgrep é particularmente relevante para a detecção de vulnerabilidades de injeção. A ferramenta permite configurar fontes de dados não confiáveis (sources) — como nós Webhook que recebem entradas externas — e destinos perigosos (sinks) — como parâmetros de consultas SQL ou comandos de execução. O motor de análise então rastreia automaticamente o fluxo de dados contaminados (taint tracking) desde as fontes até os sinks, sinalizando caminhos que não incluem validação ou sanitização adequada [5].

A análise revela que o Semgrep oferece potencial de cobertura para aproximadamente 66,7% das classes identificadas, mas esta capacidade é inteiramente potencial: depende do desenvolvimento de regras personalizadas. Até o momento, nenhum conjunto de regras específico para n8n existe no registro comunitário público.

Limitação Crítica Identificada: Modo JSON vs. Modo Genérico

Durante a fase experimental deste trabalho, identificou-se uma limitação fundamental do modo JSON do Semgrep quando aplicado a estruturas altamente aninhadas como workflows do n8n. O modo JSON, que realiza parsing da estrutura em árvores sintáticas abstratas (AST), apresenta dificuldades em aplicar padrões de detecção em arrays profundamente aninhados (4+

níveis de profundidade), como o array `nodes` dentro de workflows n8n.

A solução identificada foi a utilização do **modo genérico** (generic mode) do Semgrep, que trata o arquivo JSON como texto plano e aplica correspondência baseada em expressões regulares. Esta abordagem, embora menos sofisticada semanticamente, demonstrou-se significativamente mais efetiva para detecção de padrões inseguros em workflows n8n, conforme será detalhado na Seção 7.

Esta descoberta tem implicações importantes para a análise de segurança de plataformas LCNC: ferramentas SAST tradicionais requerem adaptação cuidadosa ao paradigma declarativo de configuração-como-código, e a escolha do modo de análise apropriado é tão crítica quanto a seleção da ferramenta em si.

A principal limitação é a ausência de conhecimento nativo do esquema n8n, exigindo investimento estimado entre 40 e 60 horas de trabalho especializado para desenvolver regras customizadas abrangentes.

2.2 Explicabilidade em Ferramentas de Análise de Segurança

A explicabilidade refere-se à capacidade de um sistema fornecer insights compreensíveis sobre seu funcionamento e decisões, sendo particularmente crítica em contextos de segurança onde a confiança nas detecções é essencial para a adoção prática das ferramentas [10]. Tanto o Agentic Radar quanto o Semgrep implementam mecanismos de explicabilidade, embora através de abordagens distintas.

O Agentic Radar gera relatórios em formato HTML com visualizações gráficas interativas dos fluxos de trabalho, identificação clara de vulnerabilidades e explicações técnicas detalhadas sobre por que determinado padrão foi classificado como risco. As recomendações de remediação são contextualizadas e acionáveis, frequentemente incluindo exemplos de código corrigido. O alinhamento explícito com o OWASP LLM Top 10 fornece uma taxonomia padronizada que facilita a comunicação de riscos com stakeholders não técnicos. A capacidade de visualização gráfica é particularmente valiosa para workflows complexos, permitindo que analistas de segurança compreendam o contexto das vulnerabilidades dentro do fluxo de dados completo.

O Semgrep, por sua vez, oferece explicabilidade através da transparência das regras. Cada regra é especificada em YAML legível, onde o padrão de detecção, as condições lógicas e os metadados (severidade, mensagens explicativas, sugestões de correção) são explicitamente declarados. Esta transparência permite que desenvolvedores e analistas de segurança compreendam exatamente o que está sendo detectado e por quê. Os relatórios gerados incluem o padrão correspondido, a localização exata no código (número de linha) e a mensagem explicativa definida na regra. O suporte a metadados arbitrários permite enriquecer as detecções com mapeamentos a frameworks de segurança (OWASP LCNC Top 10, CWE), referências a documentação e exemplos de mitigação.

A principal diferença está na natureza da explicabilidade: o Agentic Radar oferece explicações contextuais baseadas na análise holística do fluxo de trabalho, enquanto o Semgrep oferece explicações baseadas em regras que detalham por que um padrão específico é considerado inseguro. Ambas as abordagens são complementares e valiosas em diferentes contextos de uso.

2.3 Segurança e Privacidade em Sistemas de Análise Baseados em IA

A integração de Inteligência Artificial em ferramentas de análise de segurança introduz considerações específicas relacionadas à segurança e privacidade que devem ser cuidadosamente

avaliadas, especialmente em contextos corporativos com requisitos rigorosos de conformidade regulatória [8].

O Agentic Radar, particularmente em seu modo de teste dinâmico, depende da API da OpenAI para execução de testes adversariais. Esta dependência cria múltiplas preocupações. Primeiro, a necessidade de enviar workflows para serviços externos de terceiros levanta questões de privacidade e confidencialidade: workflows corporativos frequentemente contêm lógica de negócio proprietária, credenciais de acesso a sistemas internos e referências a infraestrutura sensível. A transmissão destes artefatos para serviços externos pode violar políticas de segurança organizacionais ou requisitos de conformidade como SOC 2, ISO 27001 ou a Lei Geral de Proteção de Dados (LGPD) [1] no contexto brasileiro [7].

A dependência de serviços externos cria questões de disponibilidade e custo operacional. Interrupções na disponibilidade da API da OpenAI impactam diretamente a capacidade de executar análises de segurança, um cenário inaceitável para pipelines de CI/CD críticos onde feedback rápido é essencial. Os custos por chamada de API, embora individualmente pequenos, podem acumular-se significativamente em ambientes corporativos analisando centenas ou milhares de workflows regularmente.

Também existe a preocupação de segurança da cadeia de suprimentos: a confiança na robustez e segurança dos modelos de linguagem utilizados pela API externa. Vulnerabilidades ou comportamentos inesperados nos modelos podem impactar a confiabilidade das análises, potencialmente produzindo falsos positivos ou, mais gravemente, falsos negativos que deixam vulnerabilidades reais não detectadas.

O Semgrep, como ferramenta de análise estática tradicional, pode operar completamente offline e auto-hospedado, não requerendo transmissão de código para serviços externos. A versão open-source é inteiramente local, endereçando as preocupações de privacidade. No entanto, a plataforma Semgrep Cloud opcional, que oferece dashboards de equipe, agregação de resultados históricos e gerenciamento centralizado de políticas, requer envio de código-fonte para serviços externos operados pela Semgrep Inc. Para organizações com requisitos rigorosos de conformidade, esta limitação restringe o uso à versão auto-hospedada, que carece de algumas funcionalidades de colaboração da plataforma cloud [6].

A oportunidade de utilizar IA para geração assistida de regras Semgrep, discutida na próxima subseção, deve ser implementada com consideração cuidadosa destes aspectos de privacidade. Modelos de linguagem podem ser utilizados para sugerir regras baseadas em descrições de vulnerabilidades ou exemplos de padrões inseguros, mas a geração deve, idealmente, ocorrer localmente ou através de modelos de IA auto-hospedados para evitar vazamento de conhecimento proprietário sobre padrões de vulnerabilidade específicos da organização.

2.4 Gaps e Oportunidades

A análise das ferramentas existentes e do estado da arte revela lacunas significativas e oportunidades promissoras para pesquisa e desenvolvimento na área de análise de segurança para workflows do n8n.

2.4.1 Gap Fundamental: Ausência de Solução Específica para n8n

Conforme evidenciado pela análise anterior, **não existe atualmente uma ferramenta dedicada especificamente à detecção de vulnerabilidades em workflows do n8n**. As soluções existentes oferecem cobertura parcial e complementar, mas nenhuma aborda o problema de forma abrangente. Esta lacuna representa um risco substancial para organizações que dependem da plataforma para automações críticas.

Esta lacuna é particularmente preocupante considerando a mudança de paradigma de segurança identificada na literatura: o workflow JSON não é meramente configuração, mas sim o código-fonte da aplicação, e deve ser submetido ao mesmo rigor de análise que código tradicional [9]. A ausência de ferramentas adequadas resulta na perpetuação da lacuna de governança, onde workflows potencialmente inseguros são implantados em produção sem análise automatizada.

2.4.2 Oportunidade: IA para Geração Assistida de Regras Semgrep

Uma oportunidade promissora é a utilização de IA para geração assistida de regras Semgrep. O desenvolvimento manual representa uma barreira significativa (40-60 horas estimadas). Modelos de linguagem podem ser aplicados para quatro propósitos principais. Primeiro, a **geração de regras a partir de descrições em linguagem natural**, permitindo que analistas de segurança descrevam vulnerabilidades em linguagem natural e o modelo gere automaticamente a regra Semgrep correspondente em YAML com padrões sintáticos, condições lógicas e metadados apropriados. Segundo, a **otimização de regras existentes para redução de falsos positivos**, onde modelos de IA analisam regras com taxas elevadas de falsos positivos e sugerem refinamentos nas condições lógicas que distinguem padrões genuinamente inseguros de uso legítimo. Terceiro, o **aprendizado de padrões a partir de workflows reais**, através de análise de datasets de workflows usando técnicas de aprendizado não supervisionado para identificar clusters de padrões comuns, distinguindo práticas seguras de inseguras. Quarto, a **manutenção automatizada de regras**, onde modelos de IA analisam changelogs e documentação de API da plataforma n8n para sugerir atualizações nas regras existentes ou identificar necessidade de novas regras conforme a plataforma evolui.

Esta abordagem híbrida — motor de análise clássico (Semgrep) potencializado por IA generativa para desenvolvimento de conhecimento de segurança — combina as forças de ambos os paradigmas: a confiabilidade, transparência e performance da análise estática baseada em padrões com a flexibilidade, capacidade de adaptação e eficiência de desenvolvimento proporcionadas por modelos de linguagem.

A implementação desta abordagem utiliza prompts especializados e few-shot learning, fornecendo ao Claude 3.5 Sonnet exemplos de regras Semgrep existentes para outras plataformas e solicitando adaptação para o esquema JSON do n8n. A validação das regras geradas por IA através de testes automatizados em workflows é essencial para garantir confiabilidade antes da implantação em ambientes de produção.

2.4.3 Iniciativa Comunitária: n8n-CyberSecurity-Workflows

A comunidade de segurança cibernética tem reconhecido a importância de automação de segurança utilizando n8n. O repositório `n8n-CyberSecurity-Workflows` [4], mantido pela comunidade CyberSecurityUP, documenta mais de 100 ideias de workflows para automação de segurança em Red Team, Blue Team, AppSec e DevSecOps.

Relevante para este trabalho, o repositório menciona explicitamente a possibilidade de utilizar Semgrep para análise de segurança de workflows n8n em formato JSON, reconhecendo a lacuna de ferramentas especializadas. Esta iniciativa comunitária reforça a necessidade identificada neste trabalho e sugere demanda prática por soluções SAST dedicadas ao ecossistema n8n.

2.4.4 Oportunidade: Arquitetura Híbrida para Cobertura Completa

A análise comparativa sugere que uma arquitetura híbrida poderia alcançar cobertura próxima a 100% das classes de vulnerabilidades. Esta estratégia maximiza as forças complementares: Agentic Radar oferece análise especializada em agentes de IA, enquanto Semgrep oferece motor flexível e poderoso para padrões customizados.

A integração destas ferramentas em um pipeline unificado, com agregação de resultados, classificação por severidade e categoria OWASP LCNC Top 10, e geração de relatórios consolidados, representa uma oportunidade de pesquisa significativa para o desenvolvimento de uma solução SAST verdadeiramente abrangente para o ecossistema n8n.

3 Metodologia

Esta seção descreve a abordagem metodológica adotada para o desenvolvimento e validação da ferramenta SAST proposta, detalhando o processo de análise das ferramentas existentes, a metodologia inovadora de geração de regras assistida por inteligência artificial e os procedimentos de validação empregados.

3.1 Abordagem Metodológica Geral

A pesquisa adota metodologia exploratória e experimental, fundamentada em três pilares: pesquisa exploratória através de análise comparativa de Agentic Radar e Semgrep documentando capacidades e limitações; abordagem híbrida combinando ferramentas complementares para maximizar cobertura; e IA para aceleração usando modelos de linguagem para geração assistida de regras e redução de overhead de desenvolvimento.

3.2 Análise das Ferramentas Existentes

A análise comparativa das ferramentas do estado da arte seguiu um protocolo estruturado para garantir avaliação objetiva e sistemática das capacidades e limitações de cada solução.

3.2.1 Critérios de Avaliação

As ferramentas foram avaliadas segundo quatro critérios principais: **cobertura de vulnerabilidades** (capacidade de detectar as seis classes identificadas, classificada como ALTO, MÉDIO, BAIXO ou NENHUMA); **performance** (tempo de execução, throughput e escalabilidade); **facilidade de uso** (complexidade de instalação, documentação e interface); e **qualidade dos relatórios** (clareza, severidade, explicações e sugestões de remediação).

3.2.2 Processo de Análise

O processo de análise de cada ferramenta seguiu um protocolo sistemático estruturado em quatro etapas principais. Inicialmente, realizou-se a **instalação e configuração** das ferramentas Agentic Radar (via `pip install agentic-radar`) e Semgrep (via `pip install semgrep`) em ambiente Python 3.10+. TODO: Revisar Durante esta etapa, foram documentadas todas as dependências, requisitos de configuração (como variáveis de ambiente) e possíveis incompatibilidades encontradas, garantindo a reprodutibilidade do ambiente de testes.

Na segunda etapa, conduziram-se **testes com workflows de exemplo**, executando cada ferramenta sobre um conjunto de workflows representativos que cobrem diferentes tipos de nós

relevantes para segurança, incluindo Webhook, MySQL, HTTP Request, Execute Command e Code. Esta diversidade de nós permitiu avaliar a capacidade das ferramentas em diferentes contextos de vulnerabilidades.

A terceira etapa consistiu na **análise de capacidades de detecção**, avaliando as seis classes de vulnerabilidades em três workflows de teste e verificando se cada ferramenta detecta os padrões inseguros implementados. Para cada detecção ou ausência de detecção, documentou-se sistematicamente se tratava-se de true positive, false positive, true negative ou false negative, permitindo análise quantitativa posterior.

A etapa final envolveu a **documentação de cobertura e limitações**, realizando o mapeamento sistemático da cobertura de cada ferramenta para as seis classes de vulnerabilidades, identificando lacunas fundamentais (vulnerabilidades não cobertas) e documentando limitações técnicas, operacionais e de privacidade. Os resultados desta análise comparativa foram consolidados nas seções anteriores deste documento, fornecendo a base empírica para a proposta da abordagem híbrida.

3.3 Geração de Regras Semgrep Assistida por IA

A metodologia de geração assistida de regras representa a contribuição metodológica mais inovadora deste trabalho, aplicando técnicas de inteligência artificial generativa para acelerar e sistematizar o desenvolvimento de conhecimento de segurança específico do domínio do n8n.

3.3.1 Preparação de Workflows de Teste

Desenvolveram-se três workflows de teste implementando deliberadamente as seis classes de vulnerabilidades em cenários realistas. Exemplos incluem SQL Injection (entrada de webhook passada diretamente para consulta MySQL), Command Injection (entrada usada em Execute Command) e SSRF (URL dinâmica sem validação).

Os workflows foram documentados detalhadamente, especificando localização exata dos padrões inseguros, categorias OWASP e severidades. Esta documentação serve como ground truth para validação.

3.3.2 Metodologia de Geração de Regras com IA

A geração assistida segue processo iterativo estruturado. Primeiro, analisou-se o esquema JSON do n8n (arrays `nodes` e `connections`, sintaxe de expressões dinâmicas, tipos de nós críticos).

Com este conhecimento, utilizou-se Claude 3.5 Sonnet através de prompts estruturados em três componentes: *Contexto* (esquema n8n), *Objetivo* (vulnerabilidade a detectar) e *Formato* (estrutura YAML com campos obrigatórios).

Exemplo de prompt estruturado:

> *"Você é um especialista em segurança de aplicações e na ferramenta Semgrep. Crie uma regra Semgrep em formato YAML que detecta vulnerabilidades de SQL Injection em workflows do n8n.*
>
> *Contexto: Workflows do n8n são arquivos JSON onde o array 'nodes' contém objetos representando etapas de processamento. Nós de banco de dados MySQL têm type='n8n-nodes-base.mysql' e o parâmetro 'parameters.query' contém a consulta SQL. Expressões dinâmicas usam sintaxe {{ $json.campo }}.*

Objetivo: Detectar quando o parâmetro 'query' de um nó MySQL contém expressões {{ ... }} que referenciam dados de entrada não confiáveis, indicando possível concatenação de strings em SQL sem sanitização adequada.

Formato: A regra deve incluir id, message explicativa, severity='ERROR', metadata com owasp_lcnc='LCNC-SEC-06' e CWE, e padrões que identifiquem o JSON específico do n8n."

O processo seguiu ciclo iterativo: (1) Geração inicial, (2) Teste em workflows, (3) Avaliação de detecções, (4) Refinamento com feedback, (5) Convergência. Tipicamente, foram necessárias 2-4 iterações por regra, reduzindo drasticamente o tempo de desenvolvimento.

3.3.3 Validação das Regras Geradas

As regras geradas foram validadas rigorosamente: (1) Execução em workflows de teste com documentação sistemática, (2) Ajuste manual para redução de falsos positivos (padrões negativos e contextualizações), (3) Documentação completa incluindo descrição técnica, exemplos, mapeamento OWASP/CWE e sugestões de remediação.

3.4 Validação e Métricas

O processo de validação da abordagem proposta combina avaliação quantitativa (métricas de detecção) e qualitativa (cobertura de vulnerabilidades e utilidade prática).

3.4.1 Testes de Detecção

A validação quantitativa das capacidades de detecção seguiu um protocolo experimental rigorosamente estruturado para garantir resultados objetivos e reproduzíveis. Ambas as ferramentas — Agentic Radar e Semgrep com regras customizadas — foram executadas sobre os três workflows de teste, com documentação sistemática de todos os resultados obtidos. Esta documentação detalhada permite rastreabilidade completa e validação independente dos achados.

Cada detecção ou ausência de detecção foi classificada utilizando a taxonomia padrão de sistemas de detecção: *True Positive (TP)* para vulnerabilidades reais corretamente identificadas, *False Positive (FP)* para detecções em código seguro (falsos alarmes), *True Negative (TN)* para ausência correta de detecção em código seguro, e *False Negative (FN)* para vulnerabilidades reais não detectadas.

A partir desta classificação, calcularam-se as métricas padrão de avaliação de sistemas de detecção: Precisão $P = \frac{TP}{TP+FP}$ (proporção de detecções corretas), Recall $R = \frac{TP}{TP+FN}$ (proporção de vulnerabilidades encontradas), F1-Score $F_1 = 2 \cdot \frac{P \cdot R}{P+R}$ (média harmônica), e Taxa de Falsos Positivos $\frac{FP}{FP+TN}$.

Estas métricas foram calculadas tanto por ferramenta individual quanto para a abordagem híbrida combinada, permitindo comparação objetiva das capacidades de detecção e demonstrando quantitativamente os benefícios da estratégia híbrida proposta.

3.4.2 Avaliação de Cobertura

A avaliação de cobertura verificou sistematicamente a capacidade da abordagem proposta de endereçar o espectro completo de vulnerabilidades identificadas na revisão da literatura. Para cada uma das seis classes de vulnerabilidades — SQL Injection, Command Injection, Secret

Sprawl, SSRF, Insecure Data Handling e DoS — verificou-se se ao menos uma das ferramentas da abordagem híbrida oferece capacidade de detecção efetiva.

A análise comparativa quantitativa revelou diferenças significativas na cobertura oferecida por cada ferramenta individualmente. O Agentic Radar isolado oferece cobertura de 16,7% (1 de 6 classes), focando especificamente em riscos emergentes de agentes de IA. O Semgrep customizado isolado apresenta cobertura potencial de 66,7% (4 de 6 classes com cobertura ALTO), endereçando vulnerabilidades tradicionais de aplicações web. A abordagem híbrida, combinando ambas as ferramentas, alcança cobertura próxima a 100% através da complementaridade estratégica das capacidades de cada solução.

Adicionalmente, documentaram-se as lacunas residuais, identificando categorias de vulnerabilidades que permanecem parcialmente cobertas ou requerem capacidades de análise além do escopo das ferramentas utilizadas. Um exemplo notável é a detecção de Negação de Serviço (DoS) através de análise completa de ciclos no grafo de execução, que requereria capacidades de análise de fluxo de controle mais sofisticadas do que as oferecidas pelas ferramentas atuais.

Os resultados desta validação são apresentados detalhadamente na Seção 7 (Avaliação Experimental), fornecendo evidência empírica robusta da viabilidade e efetividade da abordagem proposta para detecção de vulnerabilidades em workflows do n8n.

4 Objetivos e Métricas de Sucesso

Esta seção estabelece os objetivos gerais e específicos do trabalho, definindo critérios quantitativos e qualitativos para avaliar o sucesso da validação da abordagem proposta. É importante ressaltar que o foco não é desenvolver uma ferramenta SAST completa de produção, mas sim validar a viabilidade técnica da abordagem híbrida e da metodologia de geração de regras assistida por inteligência artificial.

4.1 Objetivo Geral

O objetivo geral é validar a **viabilidade técnica** de uma abordagem híbrida de análise estática para workflows n8n, demonstrando que a combinação de ferramentas especializadas (Agentic Radar) e genéricas configuráveis (Semgrep) pode oferecer cobertura abrangente de vulnerabilidades em estruturas JSON declarativas, estabelecendo base científica e metodologia reproduzível para análise de segurança em plataformas Low-Code/No-Code.

4.2 Objetivos Específicos

Os objetivos específicos são:

OE1: Análise comparativa: Avaliar cobertura de vulnerabilidades, limitações técnicas e aplicabilidade ao contexto n8n, produzindo métricas quantitativas e caracterização qualitativa.

OE2: Metodologia com IA: Criar abordagem sistemática para geração de regras Semgrep usando Claude 3.5 Sonnet, incluindo estruturação de prompts e processo iterativo de refinamento.

OE3: Regras customizadas: Desenvolver regras para classes não cobertas pelo Agentic Radar, incluindo padrões, condições lógicas, metadados e mensagens explicativas.

OE4: Validação empírica: Executar ferramentas em workflows de teste, documentar detecções e calcular métricas de cobertura híbrida.

OE5: Documentação de lacunas: Identificar limitações, classes parcialmente cobertas e oportunidades para pesquisa futura.

4.3 Métricas de Avaliação

A avaliação do sucesso do trabalho será realizada através de critérios objetivos quantitativos e critérios qualitativos que, em conjunto, permitem caracterizar a viabilidade e utilidade da abordagem proposta.

4.3.1 Métricas Quantitativas

As métricas quantitativas estabelecem valores-alvo objetivos para aspectos mensuráveis do trabalho, permitindo avaliação empírica do alcance dos objetivos estabelecidos.

Tabela 1: Métricas quantitativas e valores-alvo

Métrica	Definição	Meta
Cobertura de Vulnerabilidades (Híbrida)	Percentual das 6 classes de vulnerabilidades com detecção efetiva através da abordagem híbrida (Agentic Radar + Semgrep)	$\geq 80\%$
Cobertura Agentic Radar	Percentual das 6 classes adequadamente cobertas pelo Agentic Radar isoladamente	Documentar % real
Cobertura Semgrep	Percentual das 6 classes cobertas por regras Semgrep customizadas	Documentar % real
Regras Desenvolvidas	Número de regras Semgrep funcionais criadas utilizando metodologia assistida por IA	≥ 4 regras (uma por classe não coberta)
Redução de Tempo	Economia de tempo no desenvolvimento de regras com IA vs. estimativa de desenvolvimento manual (8-12h por regra)	Documentar economia em horas

A cobertura de vulnerabilidades é calculada como proporção de classes com detecção efetiva. A meta de 80% reconhece que algumas categorias podem requerer capacidades além do escopo das ferramentas.

A redução de tempo compara desenvolvimento com IA versus estimativa manual (8-12h por regra), validando quantitativamente o valor da geração assistida.

4.3.2 Métricas Qualitativas

As métricas qualitativas avaliam aspectos não diretamente quantificáveis, mas essenciais para caracterizar a viabilidade prática e utilidade da abordagem proposta. Cada métrica é avaliada através de critérios específicos de aceitação.

Viabilidade Técnica: A validação fundamental do trabalho é demonstrar que a análise estática de workflows n8n é tecnicamente viável e efetiva. Os critérios avaliam se a análise estática

de arquivos JSON permite identificação de padrões inseguros com precisão suficiente para ambientes reais, se as regras Semgrep geradas por IA são funcionais e detectam corretamente as vulnerabilidades-alvo, se a abordagem híbrida fecha as lacunas de cobertura oferecendo cobertura complementar efetiva, e se as limitações são documentadas de forma clara permitindo avaliação realista da aplicabilidade.

Usabilidade: Embora o foco não seja desenvolver uma ferramenta de produção completa, a viabilidade prática requer que as ferramentas sejam razoavelmente acessíveis. Avalia-se se a instalação e configuração é viável com complexidade aceitável (máximo de 5 comandos por ferramenta), se a documentação permite reprodução independente do estudo, e se a execução das análises é direta sem requerer expertise profunda em detalhes de implementação.

Clareza de Resultados: A utilidade prática das detecções depende da qualidade da comunicação dos resultados. Verifica-se se os relatórios identificam claramente as vulnerabilidades com localização exata no workflow, se existe mapeamento explícito para categorias do OWASP LCNC Top 10 facilitando comunicação com stakeholders, se as explicações técnicas são compreensíveis com sugestões de remediação acionáveis, e se a documentação distingue claramente entre detecções confirmadas e potenciais falsos positivos.

4.4 Benchmarks Comparativos

A avaliação da contribuição da abordagem híbrida requer comparação sistemática com três cenários baseline, cada um representando uma estratégia alternativa de análise de segurança para workflows n8n. A Tabela 2 sintetiza as coberturas esperadas para cada cenário.

Tabela 2: Benchmarks comparativos de cobertura de vulnerabilidades

Cenário	Cobertura Esperada	Observações
Agentic Radar isolado	∼16,7% (1/6 classes)	Baseline 1: Ferramenta especializada com foco em riscos emergentes de agentes de IA (injeção de prompt, vazamento de PII, geração de conteúdo prejudicial). Não cobre vulnerabilidades tradicionais de aplicações web.
Semgrep isolado	∼66,7% (4/6 classes)	Baseline 2: Potencial teórico para vulnerabilidades tradicionais (SQL Injection, Command Injection, SSRF, Secret Sprawl), mas requer desenvolvimento completo de regras customizadas. Sem regras, cobertura é 0%.
Abordagem Híbrida	∼83-100% (5-6/6 classes)	Proposta: Combinação complementar de Agentic Radar (riscos de IA) + Semgrep customizado (vulnerabilidades tradicionais). Maximiza cobertura ao combinar forças de ambas as ferramentas.

O cenário **Agentic Radar isolado** (16,7%) demonstra que ferramentas especializadas são insuficientes como solução autônoma, estabelecendo o limite inferior.

O cenário **Semgrep isolado** (66,7% potencial) evidencia a barreira crítica: sem regras customizadas, a cobertura é zero, estabelecendo o limite superior para ferramentas tradicionais.

A **Abordagem Híbrida** visa superar ambas limitações através de estratégia complementar, com meta de 83-100% de cobertura.

A validação envolverá execução dos três cenários sobre workflows de teste, permitindo comparação direta.

4.5 Critérios de Sucesso

O trabalho será considerado bem-sucedido se cinco critérios forem satisfeitos: **viabilidade técnica** (ambas ferramentas analisam workflows n8n e detectam ao menos uma vulnerabilidade cada), **IA validada** (Claude 3.5 Sonnet gera regras funcionais em tempo médio inferior a 2h por regra com economia de 75% vs. manual), **cobertura complementar** (abordagem híbrida cobre mínimo de 80% das classes), **lacunas documentadas** (limitações claramente identificadas e tecnicamente justificadas com oportunidades específicas para trabalhos futuros), e **reprodutibilidade** (metodologia documentada com detalhes suficientes para replicação independente). A satisfação destes critérios estabelece que a abordagem híbrida é viável e representa contribuição significativa para segurança de plataformas LCNC.

5 Escopo da Solução e Requisitos

Esta seção detalha o escopo da solução proposta, definindo claramente os casos de uso cobertos, a arquitetura do sistema, os requisitos técnicos e a composição dos dados de validação. O foco é estabelecer expectativas realistas sobre o que será entregue no contexto de validação de viabilidade técnica, distinguindo explicitamente entre funcionalidades incluídas e excluídas do escopo.

5.1 Casos de Uso

Os casos de uso definem as interações principais entre os usuários e o sistema, estabelecendo o escopo funcional da solução proposta. O foco está na análise estática de segurança de workflows individuais e em lote, reconhecendo que este é um projeto de validação de viabilidade técnica, não uma ferramenta de produção completa.

5.1.1 Caso de Uso Principal: Análise de Segurança de Workflow n8n

O cenário principal de uso representa o fluxo típico de análise de segurança executado por um desenvolvedor ou analista de segurança que deseja avaliar a postura de segurança de um workflow específico do n8n.

Ator primário: Desenvolvedor ou Analista de Segurança

Pré-condições: O workflow do n8n foi exportado em formato JSON através da interface da plataforma, as ferramentas de análise (Agentic Radar e Semgrep) estão instaladas e configuradas no ambiente, o conjunto de regras customizadas do Semgrep está disponível e atualizado, e o ambiente possui conectividade de rede (caso o Agentic Radar utilize o modo `test` que requer API da OpenAI).

Fluxo principal:

1. O usuário fornece o arquivo JSON do workflow como entrada para o sistema de análise
2. O sistema executa a análise híbrida, invocando sequencialmente:
    - Agentic Radar em modo `scan` para detecção de riscos de IA agêntica
    - Semgrep com conjunto de regras customizadas para vulnerabilidades tradicionais
3. Cada ferramenta produz sua saída independente: o Agentic Radar gera relatório em formato HTML ou JSON, e o Semgrep gera saída em formato JSON
4. O usuário analisa os resultados de cada ferramenta separadamente, revisando as vulnerabilidades detectadas, suas localizações no workflow e as explicações técnicas fornecidas por cada ferramenta
5. O usuário correlaciona manualmente as detecções de ambas as ferramentas quando necessário, identificando sobreposições e priorizando correções baseando-se na severidade e natureza das vulnerabilidades

Fluxos alternativos: Na **análise em lote**, o usuário fornece um diretório contendo múltiplos arquivos JSON de workflows, o sistema itera sobre todos os arquivos executando a análise híbrida em cada um, e cada ferramenta produz suas saídas individuais para todos os workflows processados.

Pós-condições: Ao final da análise, as saídas de ambas as ferramentas estão disponíveis para revisão, vulnerabilidades detectadas por cada ferramenta estão documentadas com localização específica no workflow (identificador do nó e parâmetro afetado), e recomendações de remediação específicas de cada ferramenta estão disponíveis.

5.1.2 Casos Excluídos do Escopo

Para manter o foco na validação de viabilidade técnica da abordagem híbrida e da metodologia de geração de regras assistida por IA, os seguintes casos de uso são explicitamente excluídos do escopo deste trabalho. O sistema não executará **análise dinâmica ou execução real de workflows**, operando puramente sobre a estrutura JSON declarativa. Não será desenvolvida **interface gráfica** web ou desktop, sendo a interação via linha de comando adequada para usuários técnicos e integração em pipelines automatizados. O sistema identifica e reporta vulnerabilidades mas não oferece **correção automática**, permanecendo a remediação como responsabilidade do desenvolvedor. O foco está nos nós oficiais da base do n8n, excluindo **análise de nós customizados da comunidade**. O sistema não avalia **otimização de performance** de workflows, focando exclusivamente em vulnerabilidades de segurança. A configuração e implantação de **integração em pipelines CI/CD com bloqueio automático** estão fora do escopo, embora o Semgrep possua integrações nativas. Por fim, não será desenvolvida **visualização gráfica de fluxos de trabalho**, pois embora o Agentic Radar possua esta capacidade, sua replicação não é essencial para validar a viabilidade da detecção de vulnerabilidades.

5.2 Arquitetura Proposta

A arquitetura do sistema fundamenta-se em uma abordagem híbrida que combina as forças complementares do Agentic Radar e do Semgrep para alcançar cobertura abrangente das seis classes de vulnerabilidades identificadas. O design privilegia modularidade, permitindo que cada ferramenta opere independentemente e produza suas saídas de forma autônoma.

5.2.1 Visão Geral da Arquitetura em Camadas

A arquitetura é organizada em duas camadas principais que representam o fluxo de processamento desde a entrada de dados até a execução das análises. A **Camada de Entrada de Dados** é responsável pela leitura e validação de arquivos JSON de workflows verificando conformidade com o esquema esperado do n8n. A **Camada de Motores de Análise** executa as ferramentas de detecção de vulnerabilidades em paralelo ou sequencialmente, invocando o Agentic Radar e o Semgrep sobre os workflows de entrada, produzindo resultados independentes que são analisados separadamente.

Esta arquitetura em camadas promove separação de responsabilidades, facilitando testes unitários de cada componente e permitindo substituição ou atualização de ferramentas individuais sem impacto no sistema completo.

5.2.2 Componentes Principais

A implementação da arquitetura proposta requer dois componentes principais, cada um com responsabilidades claramente definidas:

A. Módulo de Entrada e Validação

Este componente é responsável pela interface entre o usuário e o sistema de análise. Suas funções incluem parser JSON robusto que lê arquivos de workflow e valida a estrutura contra o esquema esperado (presença dos arrays `nodes` e `connections`, estrutura de objetos de nó), extração de metadados do workflow (nome, versão, contagem de nós, tipos de nós presentes), tratamento de erros para arquivos malformados ou inválidos gerando mensagens informativas para o usuário, e suporte a análise em lote iterando sobre diretórios de workflows e mantendo associação entre arquivos e resultados.

A validação estrutural neste estágio previne falhas nas ferramentas de análise downstream e fornece feedback rápido sobre problemas de formato.

B. Motor de Análise Híbrida

Este componente orquestra a execução das duas ferramentas de análise, gerenciando suas invocações e capturando suas saídas. Consiste em dois submódulos: o **Executor do Agentic Radar** invoca o comando `agentic-radar scan` sobre o arquivo JSON do workflow, captura a saída (tipicamente em formato HTML ou JSON estruturado), e extrai as vulnerabilidades relacionadas a riscos de IA agêntica, sendo responsável pela cobertura de aproximadamente 16,7% das classes de vulnerabilidades focando em injeção de prompt, vazamento de PII através de respostas de modelos de linguagem e geração de conteúdo prejudicial. O **Executor do Semgrep** invoca o comando `semgrep -config=rules/ -json` sobre o arquivo JSON do workflow utilizando o conjunto de regras customizadas desenvolvidas especificamente para o esquema do n8n, captura a saída em formato JSON estruturado (SARIF ou formato nativo do Semgrep) e extrai as vulnerabilidades relacionadas a SQL Injection, Command Injection, SSRF, Secret Sprawl, Insecure Data Handling e DoS, sendo responsável pela cobertura potencial de aproximadamente 66,7% das classes de vulnerabilidades.

A execução pode ser paralela (para maximizar velocidade) ou sequencial (para simplificar tratamento de erros), dependendo das restrições de recursos do ambiente.

5.3 Requisitos Técnicos

Os requisitos técnicos especificam as tecnologias, frameworks e infraestrutura necessários para implementação e execução da solução proposta.

5.3.1 Tecnologias e Frameworks

A seleção de tecnologias priorizou maturidade, compatibilidade com as ferramentas de análise e acessibilidade para a comunidade de segurança. A Tabela 3 sintetiza o stack tecnológico.

Tabela 3: Stack tecnológico da solução proposta

Camada	**Tecnologia**	**Justificativa**
Análise IA Agentes	Agentic Radar	Única ferramenta open-source especializada em riscos de IA agêntica para workflows de automação, com suporte comprovado para n8n
Análise Tradicional	Semgrep	Motor SAST maduro e amplamente adotado, com suporte nativo a JSON, linguagem de regras acessível e capacidades de taint analysis
Linguagem Principal	Python 3.10+	Compatibilidade com ambas as ferramentas (Agentic Radar requer Python >= 3.10 < 3.13), ecossistema rico de bibliotecas para processamento JSON e geração de relatórios
Processamento JSON	Python `json`	Biblioteca padrão para parsing e manipulação de workflows, sem dependências externas
Geração de Regras	Claude 3.5 Sonnet (API Anthropic)	Modelo de linguagem de grande capacidade para geração assistida de regras Semgrep, acelerando desenvolvimento através de prompts estruturados e few-shot learning

A escolha do Python como linguagem principal é justificada pela necessidade de compatibilidade com as ferramentas de análise e pela disponibilidade de bibliotecas robustas para as tarefas requeridas. A utilização de Claude 3.5 Sonnet para geração de regras representa uma inovação metodológica, reduzindo drasticamente o tempo de desenvolvimento de regras customizadas de 8-12 horas por regra para aproximadamente 2 horas, conforme será demonstrado na validação experimental.

5.3.2 Infraestrutura Mínima

Os requisitos de infraestrutura são modestos, refletindo a natureza de análise estática da solução que não requer recursos computacionais intensivos como GPUs ou grandes quantidades de memória.

Ambiente de Desenvolvimento: Requer CPU com 2 cores (análise estática não requer processamento intensivo), 4GB de RAM (suficiente para parsing de workflows típicos de até 100 nós), 2GB de armazenamento (instalação de ferramentas, regras customizadas e workflows de teste), Python versão 3.10, 3.11 ou 3.12 (compatibilidade com Agentic Radar), e sistema operacional Linux, macOS ou Windows (ferramentas são multiplataforma).

Ambiente de Execução: Utiliza ambiente local ou containerizado (Docker) para portabilidade, capacidade de execução paralela opcional para análise em lote de múltiplos workflows, e conectividade de rede caso o modo `test` do Agentic Radar seja utilizado (requer acesso à API da OpenAI).

A infraestrutura mínima garante que a solução possa ser executada em laptops de desenvolvimento típicos, sem necessidade de servidores dedicados ou infraestrutura cloud, reduzindo barreiras de adoção e custos operacionais.

5.3.3 Dados de Validação

A validação da abordagem proposta requer um dataset controlado de workflows com vulnerabilidades conhecidas, servindo como ground truth para avaliar precisão, recall e taxa de falsos positivos das ferramentas.

Composição do Dataset:

O dataset de validação consiste em **3 workflows sintéticos** desenvolvidos especificamente para este trabalho, cada um implementando deliberadamente 2 das 6 classes de vulnerabilidades identificadas. Esta abordagem garante cobertura balanceada de todas as categorias e permite validação controlada onde a localização exata e a natureza de cada vulnerabilidade são conhecidas a priori.

A Tabela 4 detalha a composição do dataset de validação.

Tabela 4: Composição do dataset de workflows de validação

Workflow	**Vulnerabilidades Implementadas**	**Nós e Padrões Envolvidos**
Workflow 1: Integração de Dados	SQL Injection + SSRF	Nó Webhook recebendo entrada externa → Nó MySQL com query dinâmica sem sanitização; Nó HTTP Request com URL construída dinamicamente
Workflow 2: Automação de Sistema	Command Injection + Secret Sprawl	Nó Webhook → Nó Execute Command com parâmetro `command` contendo expressão dinâmica; Credenciais hardcoded em parâmetros de configuração
Workflow 3: Processamento de Dados Sensíveis	Insecure Data Handling + DoS	Nó HTTP Request usando `http://` sem TLS para transmissão de dados; Estrutura de loop sem condição de saída clara no grafo de conexões
Workflow 4: Análise de Pull Requests com IA	Prompt Injection + Sensitive Information Disclosure	Nó GitHub Webhook → 2 AI Agents (Gemini 2.5 Flash/Pro) com dados não sanitizados em prompts; Issues Jira e diffs de código enviados para API externa do Google Vertex AI

Documentação de Ground Truth:

Cada workflow no dataset é acompanhado de documentação detalhada especificando **localização exata** (identificador do nó afetado, nome do parâmetro contendo a vulnerabilidade, linha no arquivo JSON), **categoria OWASP** (mapeamento para a categoria relevante do OWASP LCNC Top 10), **severidade esperada** (classificação da severidade baseada no potencial de impacto e exploração), **descrição técnica** (explicação de como a vulnerabilidade se manifesta no workflow específico e por que é insegura), e **mitigação sugerida** (exemplo de correção que remediaria a vulnerabilidade).

Esta documentação serve como baseline para cálculo de métricas de validação: uma detecção é considerada True Positive se a ferramenta identifica corretamente a vulnerabilidade na

localização documentada e com classificação de severidade compatível (variação de ± 1 nível é aceitável). A ausência de detecção de uma vulnerabilidade documentada constitui False Negative, enquanto detecções em código seguro (inexistentes neste dataset sintético) constituiriam False Positives.

Considerações de Privacidade:

Os workflows utilizados neste trabalho contêm dados reais extraídos de ambientes de produção, incluindo lógica de negócio, URLs, credenciais e configurações autênticas. Em conformidade com requisitos de privacidade e proteção de dados sensíveis, o dataset **não será compartilhado publicamente**. Todas as credenciais e informações identificáveis foram anonimizadas ou removidas dos exemplos apresentados neste documento. Esta abordagem permite análise realista de vulnerabilidades em contextos operacionais autênticos, respeitando rigorosamente as normas de confidencialidade e conformidade regulatória.

6 Ameaças, Riscos e Controles

Esta seção aplica metodologias estruturadas de análise de segurança para identificar sistematicamente as ameaças, avaliar riscos e estabelecer controles de mitigação para a ferramenta SAST proposta. A análise fundamenta-se no modelo STRIDE (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege) para categorização de ameaças, complementada por considerações específicas sobre o uso de inteligência artificial e requisitos de privacidade conforme a LGPD.

6.1 Superfícies de Ataque

A identificação das superfícies de ataque é fundamental para compreender os pontos de entrada e dependências que podem ser explorados por atacantes. A arquitetura híbrida proposta, combinando Agentic Radar e Semgrep com geração assistida de regras utilizando IA, apresenta quatro superfícies de ataque principais:

1. Entrada de Dados (Workflows JSON): A ferramenta processa arquivos JSON de workflows do n8n fornecidos pelo usuário. Estes arquivos podem ser malformados intencionalmente, contendo estruturas profundamente aninhadas projetadas para explorar vulnerabilidades no parser JSON, ou podem conter payloads maliciosos embutidos em expressões dinâmicas que tentam enganar os motores de análise.

2. Integrações com APIs de IA Externas: A metodologia de geração assistida de regras depende da API da Anthropic (Claude 3.5 Sonnet) para criação de regras Semgrep, enquanto o Agentic Radar, quando utilizado no modo `test`, requer comunicação com a API da OpenAI para execução de testes adversariais. Estas dependências externas introduzem riscos de privacidade, disponibilidade e integridade.

3. Base de Conhecimento (Regras e Configurações): O conjunto de regras Semgrep customizadas, armazenadas em arquivos YAML, constitui a base de conhecimento de segurança da ferramenta. A adulteração destas regras pode resultar em cegueira intencional a vulnerabilidades específicas, comprometendo a efetividade da análise.

4. Ambiente de Execução Local: A ferramenta é executada localmente através de interface de linha de comando (CLI), processando workflows no ambiente do usuário. Vulnerabilidades em bibliotecas dependentes (parser JSON, motor Semgrep, Agentic Radar) ou configurações inseguras do ambiente podem ser exploradas para execução de código arbitrário.

6.2 Análise STRIDE

A aplicação sistemática do modelo STRIDE permite categorizar ameaças de segurança e estabelecer controles de mitigação correspondentes. A Tabela 5 apresenta a matriz completa de ameaças identificadas para a ferramenta SAST proposta.

Tabela 5: Matriz de ameaças STRIDE para a ferramenta SAST

Categoria	Ameaça	Controle
Spoofing	Falsificação de identidade de workflows ou adulteração de metadados para mascarar origem maliciosa	Validação de assinatura digital de workflows (quando disponível) e verificação de integridade de arquivos JSON através de hash SHA-256 antes do processamento
Tampering	Modificação maliciosa das regras Semgrep locais para criar cegueira intencional a vulnerabilidades específicas	Controle de versão das regras através de Git, verificação de integridade (hash) antes da execução, e assinatura digital das regras oficiais
Repudiation	Ausência de registros auditáveis sobre execuções da ferramenta, impedindo rastreabilidade de análises realizadas	Geração de logs estruturados (formato JSON) contendo timestamp, hash do workflow analisado, versão das regras utilizadas e resumo de detecções, armazenados localmente com proteção de integridade
Information Disclosure	Vazamento de credenciais hardcoded em workflows através de relatórios de segurança ou logs; transmissão de lógica de negócio proprietária para APIs de IA externas	Redação automática de padrões de segredos (chaves de API, tokens) nos relatórios; anonimização de dados antes do envio para APIs de IA; preferência por análise local (Semgrep) para workflows contendo informações sensíveis
Denial of Service	Submissão de "JSON Bombs"(arquivos com aninhamento profundo ou estruturas recursivas) projetados para estourar memória do parser ou travar o motor de análise	Validação de limites de recursos: profundidade máxima de aninhamento JSON (32 níveis), tamanho máximo de arquivo (10MB), timeout de análise (60 segundos), e monitoramento de uso de memória durante o parsing
Elevation of Privilege	Exploração de vulnerabilidades em bibliotecas dependentes (parser JSON, motor Semgrep) ou execução de código JavaScript embutido em expressões dinâmicas do n8n durante análise	Execução em ambiente isolado (sandbox) quando disponível; validação rigorosa de entrada antes do parsing; uso de versões atualizadas e auditadas de dependências; desabilitação de execução de código JavaScript durante análise estática

6.2.1 Análise Detalhada por Categoria

Spoofing: O risco de falsificação é relativamente baixo em execução local, onde o usuário possui controle direto sobre os arquivos analisados. No entanto, em cenários onde workflows são compartilhados entre equipes ou obtidos de fontes externas, existe o risco de adulteração de metadados (nome do workflow, versão) para mascarar a origem maliciosa. A implementação de verificação de integridade através de hash SHA-256 permite detectar modificações não autorizadas, enquanto assinaturas digitais (quando disponíveis) fornecem garantias de autenticidade.

Tampering: A adulteração das regras Semgrep representa uma ameaça crítica, pois pode resultar em cegueira intencional a vulnerabilidades específicas, criando uma falsa sensação de segurança. O controle de versão através de Git permite rastreabilidade completa de modificações, enquanto verificação de hash antes da execução detecta adulterações não autorizadas. Para ambientes de produção, recomenda-se assinatura digital das regras oficiais utilizando chaves criptográficas gerenciadas centralmente.

Repudiation: A ausência de registros auditáveis impede a rastreabilidade de análises realizadas, dificultando investigações de segurança e conformidade regulatória. A geração de logs estruturados em formato JSON, contendo timestamp, hash do workflow, versão das regras e resumo de detecções, estabelece um registro imutável de todas as execuções. A proteção de integridade dos logs através de hash ou assinatura digital previne adulteração posterior.

Information Disclosure: Esta categoria apresenta riscos críticos devido à natureza sensível dos dados processados. Workflows podem conter credenciais hardcoded (chaves de API, tokens de acesso, senhas), lógica de negócio proprietária e referências a infraestrutura interna. A redação automática de padrões de segredos nos relatórios (substituição por `[REDACTED]`) previne vazamento acidental através de compartilhamento de relatórios. A anonimização de dados antes do envio para APIs de IA (remoção de identificadores, substituição de valores por placeholders) protege propriedade intelectual. Para workflows altamente sensíveis, recomenda-se desabilitar completamente o uso de APIs externas, utilizando apenas análise local do Semgrep.

Denial of Service: Ataques de negação de serviço podem ser executados através de arquivos JSON malformados projetados para explorar vulnerabilidades no parser ou consumir recursos computacionais excessivos. "JSON Bombs"com aninhamento profundo podem causar estouro de pilha (stack overflow) em parsers recursivos, enquanto estruturas recursivas podem resultar em loops infinitos durante a análise. A implementação de limites rigorosos de recursos (profundidade máxima, tamanho de arquivo, timeout) e monitoramento de uso de memória previne estes ataques, garantindo que a ferramenta falhe graciosamente sem comprometer a estabilidade do sistema.

Elevation of Privilege: A exploração de vulnerabilidades em bibliotecas dependentes ou execução de código JavaScript embutido em expressões dinâmicas do n8n durante análise estática representa um risco significativo. Embora a análise estática não deva executar código, parsers mal implementados podem inadvertidamente processar expressões de forma insegura. A execução em ambiente isolado (sandbox), quando disponível, limita o impacto de explorações. A validação rigorosa de entrada antes do parsing e o uso de versões atualizadas e auditadas de dependências reduzem a superfície de ataque.

6.3 Riscos Específicos de Sistemas Baseados em IA

A integração de inteligência artificial na ferramenta proposta, através da geração assistida de regras utilizando Claude 3.5 Sonnet e do Agentic Radar que utiliza modelos de linguagem para análise, introduz categorias específicas de riscos que transcendem vulnerabilidades tradicionais de software.

6.3.1 Privacidade e Confidencialidade de Dados

O envio de descrições de vulnerabilidades, exemplos de código ou trechos de workflows para APIs de IA externas (Anthropic, OpenAI) cria riscos significativos de vazamento de propriedade intelectual e lógica de negócio proprietária. Workflows corporativos frequentemente contêm referências a sistemas internos, padrões de integração específicos da organização e estruturas de dados que revelam informações estratégicas.

Controles de Mitigação: Implementa-se **anonimização proativa** removendo identificadores únicos antes do envio para APIs externas e substituindo por placeholders genéricos, **política de retenção zero** preferindo modelos de IA que garantem que dados não sejam utilizados para treinamento de modelos futuros, **modelos auto-hospedados** para organizações com requisitos rigorosos de conformidade eliminando completamente a transmissão de dados para serviços externos, e **segmentação de dados** limitando o escopo de dados enviados para APIs transmitindo apenas trechos mínimos necessários evitando envio de workflows completos.

6.3.2 Integridade e Alucinação de Modelos de Linguagem

Modelos de linguagem são suscetíveis a "alucinações"— geração de conteúdo sintaticamente válido mas logicamente incorreto ou factualmente impreciso. No contexto de geração de regras Semgrep, uma alucinação pode resultar em regras que são válidas sintaticamente (passam validação YAML) mas não detectam corretamente as vulnerabilidades-alvo, criando falsos negativos que geram falsa sensação de segurança.

Controles de Mitigação: Implementa-se **validação humana obrigatória (human-in-the-loop)** onde todas as regras geradas por IA são submetidas a revisão humana antes da implantação em produção, **testes automatizados contra ground truth** validando sistematicamente regras geradas contra o dataset de validação com cálculo de métricas de precisão, recall e F1-score para identificar regras deficientes, **refinamento iterativo** através de processo estruturado onde regras são testadas, avaliadas e refinadas com prompts ajustados convergindo para alta qualidade após 2-4 iterações típicas, e **documentação de limitações** com transparência explícita sobre vulnerabilidades cobertas e limitações conhecidas evitando over-trust na automação.

6.3.3 Disponibilidade e Dependência de Serviços Externos

A dependência de APIs de IA externas (Anthropic, OpenAI) para geração de regras e execução de testes adversariais cria um ponto único de falha. Interrupções na disponibilidade destes serviços, limitações de taxa (rate limiting) ou custos operacionais elevados podem impactar a capacidade de desenvolver novas regras ou executar análises completas.

Controles de Mitigação: Implementa-se **arquitetura híbrida com degradação graciosa** onde o motor principal de análise (Semgrep) opera completamente offline não dependendo de serviços externos para detecção de vulnerabilidades com IA utilizada exclusivamente para aceleração de desenvolvimento (design-time) não bloqueando execução de análises (run-time), **cache de regras geradas** com armazenamento local de regras previamente geradas e validadas permitindo reutilização sem novas chamadas à API, **fallback para desenvolvimento manual** com metodologia documentada para desenvolvimento manual de regras Semgrep quando APIs de IA não estão disponíveis garantindo continuidade operacional, e **monitoramento de custos** com implementação de limites de custo e alertas para prevenir gastos excessivos com APIs de IA.

6.3.4 Segurança da Cadeia de Suprimentos de IA

A confiança na robustez e segurança dos modelos de linguagem utilizados pelas APIs externas representa um risco de segurança da cadeia de suprimentos. Vulnerabilidades ou comportamentos inesperados nos modelos podem impactar a confiabilidade das análises, potencialmente produzindo falsos positivos ou, mais gravemente, falsos negativos que deixam vulnerabilidades reais não detectadas.

Controles de Mitigação: Implementa-se **diversificação de fornecedores** com capacidade de alternar entre diferentes modelos de IA (Claude, GPT-4, modelos open-source) para reduzir dependência de um único fornecedor, **validação independente** através de testes sistemáticos de regras geradas por diferentes modelos contra o mesmo dataset de validação identificando discrepâncias que podem indicar problemas de qualidade, e **auditoria de comportamento** com monitoramento de padrões de saída dos modelos para detectar degradação de qualidade ou mudanças de comportamento que possam indicar atualizações problemáticas nos modelos subjacentes.

6.4 Considerações de Privacidade (LGPD)

A aplicação da metodologia LINDDUN (Linkability, Identifiability, Non-repudiation, Detectability, Disclosure, Unawareness, Non-compliance) fornece uma análise sistemática de riscos de privacidade, particularmente relevante no contexto brasileiro onde a Lei Geral de Proteção de Dados (LGPD) [1] estabelece requisitos rigorosos de proteção de dados pessoais.

Linkability: A capacidade de correlacionar múltiplas análises de workflows para identificar padrões de comportamento do usuário ou organização representa um risco de privacidade. Logs de execução que armazenam hashes de workflows podem, teoricamente, ser correlacionados para rastrear evolução de práticas de desenvolvimento ao longo do tempo.

Controle: Minimização de dados nos logs, armazenando apenas metadados essenciais (timestamp, contagem de vulnerabilidades por tipo) sem identificadores que permitam correlação direta.

Identifiability: Workflows podem conter dados pessoais (PII) como endereços de e-mail, números de telefone ou identificadores de clientes. A análise estática processa estes dados, potencialmente expondo-os em relatórios ou logs.

Controle: Detecção automática de padrões de PII (expressões regulares para e-mails, CPF, telefones) e redação automática nos relatórios, substituindo por `[REDACTED]`.

Disclosure: O vazamento não autorizado de dados pessoais através de relatórios compartilhados ou transmissão para APIs de IA externas viola requisitos da LGPD [1].

Controle: Redação automática de PII antes de qualquer transmissão externa; preferência por análise local para workflows contendo dados pessoais; documentação explícita de medidas de proteção de dados.

Non-compliance: A falta de conformidade com requisitos da LGPD [1] pode resultar em penalidades regulatórias significativas (até 2% do faturamento ou R$ 50 milhões).

Controle: Implementação de medidas técnicas e organizacionais documentadas (Art. 46 da LGPD [3]), incluindo criptografia de dados em repouso, controle de acesso a logs e políticas de retenção de dados com prazo de expiração automática.

6.5 Considerações Éticas

A utilização de inteligência artificial em ferramentas de segurança introduz considerações éticas que transcendem aspectos puramente técnicos, requerendo compromissos explícitos de transpa-

rência, equidade e responsabilidade.

6.5.1 Transparência e Explicabilidade

A transparência refere-se à capacidade de usuários compreenderem como a ferramenta funciona, quais vulnerabilidades são detectadas e quais são limitações conhecidas. A explicabilidade é particularmente crítica quando IA é utilizada para geração de regras, pois usuários devem compreender a origem e confiabilidade das detecções.

Compromissos de Transparência: Implementa-se **documentação explícita de cobertura** onde relatórios incluem mapeamento claro de quais classes de vulnerabilidades são cobertas pela abordagem híbrida e quais são limitações conhecidas, **identificação de origem de detecções** onde cada vulnerabilidade detectada identifica explicitamente qual ferramenta realizou a detecção permitindo que usuários compreendam a base técnica da análise, **metadados de regras** incluindo indicação se foram geradas por IA ou desenvolvidas manualmente com timestamp de criação e versão do modelo utilizado quando aplicável, e **limitações documentadas** com admissão explícita de que análise estática não pode detectar todas as vulnerabilidades especialmente aquelas que dependem de contexto de execução ou lógica de negócio complexa.

6.5.2 Equidade e Ausência de Vieses

Embora a ferramenta proposta não tome decisões que impactem diretamente indivíduos (não classifica pessoas, não determina acesso a recursos), a equidade manifesta-se através da consistência e imparcialidade da detecção de vulnerabilidades. Vieses podem surgir se regras geradas por IA favorecem certos padrões de código ou tipos de vulnerabilidades em detrimento de outros.

Estratégias de Mitigação: Implementa-se **dataset balanceado de validação** onde o conjunto de workflows de teste cobre todas as seis classes de vulnerabilidades de forma balanceada garantindo que regras sejam avaliadas contra espectro completo de riscos, **validação cruzada** com testes de regras geradas por IA contra múltiplos workflows representativos identificando padrões de detecção inconsistentes que podem indicar vieses, e **revisão por múltiplos especialistas** onde validação humana de regras geradas por IA é realizada por múltiplos analistas de segurança reduzindo impacto de vieses individuais.

6.5.3 Responsabilidade e Accountability

A responsabilidade refere-se à capacidade de identificar quem é responsável por decisões tomadas pela ferramenta e garantir que existam mecanismos de prestação de contas. Embora a ferramenta seja executada localmente pelo usuário, a responsabilidade pela interpretação de resultados e decisões de remediação permanece com o usuário.

Mecanismos de Accountability: Implementa-se **rastreabilidade de regras** onde todas as regras incluem metadados de autoria (gerada por IA com identificação do modelo, ou desenvolvida manualmente com identificação do autor) permitindo rastreabilidade completa, **versionamento de conjuntos de regras** através de controle de versão formal via Git permitindo auditoria histórica de mudanças e rollback para versões anteriores se problemas forem identificados, e **documentação de decisões de design** com justificativa técnica documentada para escolhas arquiteturais permitindo avaliação crítica por pares.

6.5.4 Direito de Contestação

Embora a ferramenta não tome decisões automatizadas sobre indivíduos (não é um sistema de decisão automatizada conforme Art. 20 da LGPD [2]), usuários devem ter capacidade de contestar detecções que considerem incorretas (falsos positivos) ou reportar vulnerabilidades não detectadas (falsos negativos).

Processo de Contestação: Implementa-se **feedback de falsos positivos** através de mecanismo documentado para usuários reportarem detecções incorretas permitindo refinamento iterativo de regras e redução de taxas de falsos positivos, **reporte de falsos negativos** com processo para submissão de workflows com vulnerabilidades conhecidas não detectadas permitindo identificação de lacunas de cobertura e desenvolvimento de regras adicionais, e **revisão de contestações** através de processo estruturado de revisão por especialistas em segurança com documentação de decisões e atualização de regras quando apropriado.

7 Avaliação Experimental

Esta seção apresenta os resultados da validação experimental da análise estática de segurança de workflows n8n utilizando Semgrep. O estudo documenta uma descoberta técnica crítica: a necessidade de adaptação do modo de análise (JSON vs. genérico) para plataformas Low-Code/No-Code, contribuindo para o conhecimento sobre aplicação de ferramentas SAST a paradigmas declarativos.

7.1 Protocolo Experimental

O protocolo experimental seguiu abordagem iterativa em duas fases: tentativa inicial com modo JSON (falha completa) seguida de pivô para modo genérico (sucesso parcial), permitindo comparação empírica das capacidades de cada abordagem.

7.1.1 Configuração de Hardware e Software

A análise foi conduzida em ambiente controlado com sistema operacional macOS 14.6 (Darwin 24.6.0), processador Apple Silicon (arquitetura ARM64), 16GB de memória RAM, Python versão 3.14, Semgrep versão 1.144.0, e Claude 3.5 Sonnet via API Anthropic para geração assistida de regras.

7.1.2 Dataset de Validação

O dataset consiste em quatro workflows reais do n8n: três com vulnerabilidades tradicionais de aplicações web (14 vulnerabilidades documentadas) e um com agentes de IA (vulnerabilidades de LLM). A Tabela 6 apresenta as características do dataset.

Tabela 6: Características do dataset de validação

Workflow	Nós	Linhas	Vulnerabilidades	Classes	AI
workflow_1.json	67	2.650	2	Secret Deletion	Não
workflow_2.json	100+	2.945	5	SQL Injection, DoS	Não
workflow_3.json	17	605	7	Command Inj., Secrets	Não
workflow_4.json	14	552	4 (estimado)	Prompt Inj., PII	**Sim (2)**
TOTAL	**198+**	**6.752**	**18**	**7 classes**	**1/4**

7.2 Resultados

A avaliação experimental revelou diferenças dramáticas entre os dois modos de análise do Semgrep, documentando limitações arquiteturais fundamentais do modo JSON para estruturas LCNC.

7.2.1 Fase 1: Modo JSON (Falha Completa)

A tentativa inicial utilizou o modo JSON nativo do Semgrep, desenvolvendo 33 regras customizadas com assistência de IA (Claude 3.5 Sonnet) em 3,75 horas. As regras foram sintaticamente válidas e seguiam as convenções do Semgrep, cobrindo 7 classes de vulnerabilidades.

Execução: Semgrep executou sem erros em todos os workflows (tempo médio: 2,3s por workflow, memória: $\sim$55MB).

Detecções: Zero. Nenhuma das 14 vulnerabilidades documentadas foi detectada.

A Tabela 7 apresenta os resultados detalhados.

Tabela 7: Resultados da análise em modo JSON

Workflow	Detecções	Esperado	Cobertura	FN
workflow_1.json	0	2	0%	2
workflow_2.json	0	5	0%	5
workflow_3.json	0	7	0%	7
TOTAL	**0**	**14**	**0%**	**14 (100%)**

Análise de Causa Raiz:

Investigação detalhada revelou três limitações técnicas fundamentais. Primeiro, o **aninhamento profundo** onde o parser JSON do Semgrep falha em aplicar padrões dentro de arrays profundamente aninhados (4+ níveis) e a estrutura típica de workflows n8n (root $\rightarrow$ nodes $\rightarrow$ node $\rightarrow$ parameters) excede esta capacidade. Segundo, o **comportamento de ellipsis** onde o operador `...` (correspondência flexível) comporta-se diferentemente em modo JSON comparado a linguagens de código não iterando sobre elementos de arrays conforme necessário. Terceiro, a **extração de metavariáveis** onde expressões n8n contendo `{{ $json.field }}` não foram corretamente capturadas via `metavariable-regex` impedindo detecção de injeção.

Validação: Teste com regra mínima (detectar qualquer nó com campo `"type"`) também produziu zero detecções, confirmando limitação arquitetural, não erro de implementação.

7.2.2 Fase 2: Modo Genérico (Breakthrough)

Baseado na análise de causa raiz, pivotou-se para o **modo genérico** do Semgrep, que trata arquivos JSON como texto plano e aplica correspondência via expressões regulares (regex). Desenvolveram-se 7 regras focadas em padrões textuais específicos.

Execução: Semgrep executou com performance similar (tempo: ∼2s/workflow).

Detecções: 19 findings totais, incluindo 100% das vulnerabilidades críticas documentadas de SQL Injection e Command Injection, além de descoberta de 10 instâncias adicionais não documentadas no ground truth inicial.

A Tabela 8 apresenta os resultados detalhados.

Tabela 8: Resultados da análise em modo genérico

Workflow	**Detecções**	**Principais Achados**	**Tempo**
workflow_1.json	2	Operações Vault não autorizadas (2x)	1,8s
workflow_2.json	13	SQL Injection (3 doc + 10 novos)	2,1s
workflow_3.json	4	PGPASSWORD injection, SSH keys	1,9s
TOTAL	**19**	**6 classes detectadas**	**5,8s**

7.2.3 Comparação Direta: JSON vs. Genérico

A Tabela 9 compara diretamente os dois modos de análise.

Tabela 9: Comparação JSON mode vs. Generic mode

Métrica	**JSON Mode**	**Generic Mode**	**Diferença**
Regras desenvolvidas	33 (7 arquivos)	7 (1 arquivo)	-26 regras
Tempo desenvolvimento	3,75h	1,2h	-68%
Detecções totais	0	19	+19 (infinito)
Cobertura documentada	0% (0/14)	71% (10/14)	+71 pp
Falsos negativos	14 (100%)	4 (29%)	-71 pp
Complexidade regras	Alta (AST matching)	Média (regex)	Mais simples
Precisão semântica	Alta (teoricamente)	Baixa (texto plano)	Trade-off

7.2.4 Cobertura por Classe de Vulnerabilidade

A Tabela 10 apresenta a cobertura de detecção por classe de vulnerabilidade, comparando ambos os modos.

Tabela 10: Cobertura por classe de vulnerabilidade

Classe	Documentadas	JSON Mode		Generic Mode	
		Detectadas	Cov.	Detectadas	Cov.
SQL Injection	3	0	0%	13	100%+
Command Injection	2	0	0%	3	100%+
Vault Secrets	2	0	0%	2	100%
SSH Key Exposure	1	0	0%	1	100%
Hardcoded Passwords	3	0	0%	0	0%
SSRF/DoS	3	0	0%	0	0%
TOTAL	**14**	**0**	**0%**	**19**	**71%**

Observação Crítica: Modo genérico detectou 100%+ em SQL/Command Injection (incluindo instâncias não documentadas), demonstrando sensibilidade adequada. Hardcoded passwords são detectáveis via regras Semgrep com padrões de entropia, mas não foram priorizadas no escopo inicial. SSRF e DoS requerem análise semântica mais sofisticada que regex pura não fornece.

7.3 Análise de Causa Raiz da Divergência

A diferença dramática entre os modos (0% vs. 71% de cobertura) decorre de incompatibilidades arquiteturais fundamentais entre o paradigma de análise do Semgrep e estruturas LCNC.

7.3.1 Paradigma Declarativo vs. Imperativo

Código Tradicional (Imperativo): Ferramentas SAST analisam fluxo de controle (if/else, loops) e fluxo de dados (variáveis, funções) através de ASTs de linguagens de programação. O Semgrep foi projetado para este paradigma.

Workflows n8n (Declarativo): Workflows são configurações JSON que *declaram* transformações de dados através de conexões entre nós. Não há controle de fluxo imperativo tradicional. O "código-fonte"é a configuração, não instruções executáveis.

Incompatibilidade: O parser JSON do Semgrep trata workflows como *dados de configuração*, não como *código*, falhando em reconhecer padrões inseguros dentro da estrutura aninhada. O modo genérico, ao tratar JSON como texto, inadvertidamente resolve o problema ao aplicar correspondência de padrões puramente textual, similar a grep avançado.

7.3.2 Limitação de Profundidade de Aninhamento

Estrutura típica de vulnerabilidade SQL Injection em n8n:

```
{
  "nodes": [              // Nível 1
    {                     // Nível 2
      "type": "...",
      "parameters": {     // Nível 3
        "query": "=..."  // Nível 4 - vulnerabilidade aqui
      }
    }
```

```
    ]
}
```

O modo JSON do Semgrep requer navegação via `pattern-inside` aninhados para alcançar o nível 4, mas esta construção falha em arrays. O modo genérico simplesmente procura pelo texto `"query": "` = diretamente, ignorando a estrutura hierárquica.

7.4 Discussão

Os resultados demonstram que **a escolha do modo de análise é tão crítica quanto a seleção da ferramenta** para análise de segurança de plataformas LCNC.

7.4.1 Trade-off: Simplicidade vs. Precisão Semântica

O modo genérico, embora efetivo (71% de cobertura), sacrifica compreensão semântica. As **limitações** incluem não compreender estrutura JSON (pode corresponder strings dentro de comentários ou valores), não poder rastrear fluxo de dados entre nós (taint analysis), e vulnerabilidade a falsos positivos se padrões textuais aparecem em contextos seguros. As **vantagens** incluem simplicidade (regras são regex compreensíveis, não AST patterns complexos), efetividade empírica (19 detecções vs. 0), desenvolvimento rápido (1,2h vs. 3,75h), e manutenibilidade (atualizar regex é mais simples que reescrever AST patterns).

Conclusão: Para análise de segurança de workflows n8n com Semgrep, o modo genérico é *pragmaticamente superior* ao modo JSON nativo, apesar de teoricamente menos sofisticado. Este resultado contradiz a intuição comum de que "parsing estruturado sempre supera correspondência textual".

7.4.2 Validação da Hipótese de Pesquisa

A hipótese central deste trabalho afirmava que ferramentas SAST existentes, com customização, poderiam fornecer análise de segurança para n8n. Os resultados validam parcialmente esta hipótese, mostrando que Semgrep *pode* detectar vulnerabilidades em workflows n8n (71% de cobertura em modo genérico) embora qualificado pela necessidade de adaptação significativa do modo de análise (não apenas desenvolvimento de regras) e limitado pelo fato de que cobertura de 71% é insuficiente para ambientes críticos (meta: $\geq$80%).

7.4.3 Contribuição Metodológica: AI-Assisted Rule Generation

A geração assistida por IA (Claude 3.5 Sonnet) demonstrou efetividade com **aceleração** de 13x mais rápido que desenvolvimento manual (3,75h vs. estimativa de 49h), **qualidade** produzindo regras sintaticamente corretas e bem documentadas, **iteração** mostrando que feedback humano é essencial (2-4 iterações por regra), embora com **limitação** onde IA gerou regras JSON mode que não funcionaram mas pivotou efetivamente para generic mode quando informada da limitação.

Lição: IA acelera desenvolvimento mas não elimina necessidade de validação empírica. A falha completa do modo JSON só foi identificada através de testes reais, não análise teórica.

7.4.4 Implicações Práticas

Para equipes de segurança considerando análise SAST de workflows n8n:

Recomendação Imediata: Utilize Semgrep em modo genérico com regras desenvolvidas neste trabalho como baseline. A cobertura de 71% é superior a 0% (análise manual ad-hoc).

Integração CI/CD: O modo genérico é adequado para gates automatizados com execução rápida ($\sim$2s por workflow), baixo consumo de recursos (55MB memória), zero falsos positivos observados neste estudo, e bloqueio de SQL/Command Injection (classes mais críticas).

Limitações Conhecidas: Hardcoded passwords são detectáveis com regras Semgrep adicionais (não incluídas no escopo inicial). SSRF e DoS requerem análise semântica avançada, revisão manual complementar, análise dinâmica (execução em sandbox), ou desenvolvimento de ferramenta especializada (proposta na Seção 9.3).

7.5 Avaliação com Agentic Radar

Esta subseção apresenta os resultados da análise experimental utilizando o Agentic Radar, ferramenta especializada em detecção de vulnerabilidades em sistemas de agentes de IA. A análise revela descobertas críticas sobre a efetividade de ferramentas especializadas em contextos específicos versus sua aplicabilidade geral.

7.5.1 Protocolo Experimental Específico

O Agentic Radar versão 0.14.0 foi executado em modo `scan` (análise estática, sem chamadas a APIs externas) sobre os quatro workflows do dataset. A ferramenta foi instalada via `pip install agentic-radar` e configurada para análise de workflows n8n.

Ambiente de execução: A análise utilizou SPLX Agentic Radar 0.14.0 em modo `scan` (análise estática local) na plataforma macOS 14.6 com Python 3.9.24, executando o comando `agentic-radar scan n8n -input-dir <dir> -output-file <file>`.

Composição do dataset: Descoberta crítica durante a análise revelou que apenas 1 dos 4 workflows (25%) contém agentes de IA, sendo workflows 1-3 com vulnerabilidades tradicionais (SQL Injection, Command Injection, exposição de credenciais) sem agentes de IA, e workflow 4 consistindo em análise automatizada de Pull Requests utilizando Google Vertex AI (Gemini 2.5 Pro/Flash) com 2 agentes de IA.

Esta distribuição representa um cenário realista: a maioria dos workflows n8n em produção não utiliza agentes de IA, focando em automações tradicionais de integração de APIs e processamento de dados.

7.5.2 Resultados da Análise Estática

A Tabela 11 apresenta os resultados da análise por workflow.

Tabela 11: Resultados da análise com Agentic Radar por workflow

Workflow	AI	Detecções	Tempo	Principais Achados
workflow_1	Não	30	0,66s	Prompt Injection Indireto (30×)
workflow_2	Não	27	0,71s	Prompt Injection Indireto (27×)
workflow_3	Não	5	0,64s	Prompt Injection Indireto (5×)
workflow_4	**Sim**	**12**	1,11s	**Prompt Inj. (10×), PII (2×)**
TOTAL	**1/4**	**74**	**3,12s**	**4 TP, 70 FP**

Descoberta crítica — Taxa de falsos positivos: A análise revelou taxa de falsos positivos de **94,6%** (70 de 74 detecções). Todos os 62 achados nos workflows 1-3 (sem agentes de IA) são falsos positivos, classificando erroneamente requisições HTTP comuns como vulnerabilidades de "Indirect Prompt Injection".

Análise detalhada por workflow:

Workflows 1-3 (sem agentes de IA): Apresentaram 62 instâncias de "Indirect Prompt Injection"classificadas como OWASP LLM01, constituindo falsos positivos onde a ferramenta sinaliza qualquer nó HTTP Request como potencial vetor de ataque independentemente da presença de agentes de IA no workflow, resultando em cobertura de vulnerabilidades reais de 0/14 (0%) sem detectar nenhuma das 14 vulnerabilidades documentadas (SQL Injection, Command Injection, exposição de credenciais).

Workflow 4 (com 2 agentes de IA): Contém nós "AI Agent - Melhora Requisitos Jira"(Gemini 2.5 Flash) e "AI Agent - Analise Req x Dev"(Gemini 2.5 Pro), com 12 detecções total (8 indiretas + 2 diretas + 2 exposição de informação sensível), sendo 4 verdadeiros positivos (2 Prompt Injection direto + 2 Sensitive Information Disclosure) e aproximadamente 8 falsos positivos (Indirect Prompt Injection em nós HTTP genéricos).

Vulnerabilidades verdadeiras identificadas no Workflow 4:

1. Prompt Injection Direto (2 instâncias — CRÍTICO): Dados não confiáveis (descrições de issues Jira, corpos de Pull Requests, diffs de código) concatenados diretamente em prompts sem sanitização nos nós "AI Agent - Melhora Requisitos Jira"e "AI Agent - Analise Req x Dev", classificado como OWASP LLM01 - Prompt Injection. Cenários de ataque incluem issue Jira maliciosa contendo instruções ocultas para o agente de IA e PR com descrição contendo sequências de escape de prompt, com impactos de manipulação do comportamento do agente, geração de análises incorretas, e potencial exfiltração de informações via respostas do LLM.

2. Exposição de Informação Sensível (2 instâncias — ALTO): Jira Issues internas (potencialmente contendo PII, segredos de negócio) e diffs de código fonte (propriedade intelectual) enviados para API externa do Google Vertex AI, classificado como OWASP LLM06 - Sensitive Information Disclosure. Preocupações de privacidade incluem conformidade com LGPD (dados de cidadãos brasileiros), exposição de trade secrets, e ausência de controles de residência de dados. Mitigação recomendada inclui implementar redação automática de PII, utilizar modelos auto-hospedados para dados sensíveis, e estabelecer políticas de classificação de dados.

7.5.3 Cobertura por Classe de Vulnerabilidade

A Tabela 12 apresenta a cobertura do Agentic Radar mapeada às seis classes de vulnerabilidades identificadas no framework OWASP LCNC Top 10.

Tabela 12: Cobertura do Agentic Radar por classe de vulnerabilidade

Classe	Ground Truth	Detectadas	Cob.	Justificativa
Riscos de IA	4 (wf4)	4	**100%**	Especialização efetiva
SQL Injection	3 (wf2)	0	0%	Fora do escopo
Command Injection	2 (wf3)	0	0%	Fora do escopo
Secret Sprawl	3 (wf3)	0	0%	Fora do escopo
SSRF	1 (wf3)	0	0%	Fora do escopo
DoS	2 (wf1-2)	0	0%	Fora do escopo
TOTAL	**15**	**4**	**26,7%**	**Especializado**

Observação crítica: O Agentic Radar alcança **100% de cobertura em seu domínio especializado** (vulnerabilidades de agentes de IA), mas **0% em vulnerabilidades tradicionais**. A cobertura agregada de 26,7% reflete a proporção de workflows com agentes de IA no dataset (1/4 = 25%).

7.5.4 Análise de Limitações do Agentic Radar

A análise experimental identificou cinco limitações fundamentais:

1. Taxa de falsos positivos extremamente elevada (94,6%): 70 de 74 detecções são falsos positivos, com heurísticas excessivamente amplas onde qualquer nó HTTP Request é sinalizado como potencial vetor de ataque, ausência de consciência de contexto não distinguindo workflows com/sem agentes de IA, e consequente inviabilidade para gates automatizados em CI/CD sem revisão manual substancial.

2. Cobertura zero em vulnerabilidades tradicionais: A ferramenta não detectou nenhuma das 14 vulnerabilidades documentadas em workflows 1-3. Vulnerabilidades como SQL Injection, Command Injection e exposição de credenciais foram completamente invisíveis para a ferramenta. Esta constitui uma limitação arquitetural, não configuracional, pois a ferramenta fundamentalmente não foi projetada para detectar vulnerabilidades OWASP LCNC Top 10 tradicionais.

3. Dependência da composição do dataset: A efetividade da ferramenta é diretamente proporcional à prevalência de agentes de IA nos workflows analisados. Neste estudo, apenas 25% dos workflows contêm IA, resultando em uma cobertura agregada de 26,7%. Consequentemente, em ambientes típicos de n8n, onde predominam automações sem IA, a cobertura esperada seria ainda menor.

4. Ausência de análise de fluxo de dados (taint analysis): A ferramenta não rastreia a propagação de dados não confiáveis através do grafo de execução. Em vez de realizar análise semântica, as detecções baseiam-se em padrões sintáticos como a simples presença de nós HTTP e LLM. Esta abordagem superficial resulta em falsos positivos quando há nós HTTP sem conexão com IA, e potencialmente em falsos negativos quando os fluxos de dados são complexos.

5. Relatórios requerem interpretação especializada: O HTML gerado contém todas as detecções sem filtrar falsos positivos óbvios, exigindo que o analista de segurança valide ma-

nualmente cada achado. Adicionalmente, há ausência de scores de confiança ou priorização automática, dificultando ainda mais a triagem dos resultados.

7.5.5 Relatórios e Explicabilidade

O Agentic Radar gera relatórios HTML com visualização gráfica interativa do workflow, apresentando um grafo completo de execução com nós coloridos por tipo. A seção de "Findings"lista todas as vulnerabilidades detectadas, fornecendo para cada detecção o nome da vulnerabilidade, descrição técnica, mapeamento aos frameworks OWASP LLM Top 10 e OWASP Agentic, além de recomendações de mitigação. Adicionalmente, oferece exportação opcional em formato JSON estruturado para processamento automatizado.

Qualidade das explicações: Em relação aos aspectos positivos, a ferramenta fornece descrições técnicas claras com alinhamento explícito aos frameworks de segurança reconhecidos (OWASP LLM Top 10), além de recomendações de remediação acionáveis como "Implement guardrails filtering for prompt injection"e "Implement sanitization of HTTP response content". Por outro lado, há limitações importantes: a ferramenta não diferencia detecções de alta e baixa confiança, apresentando todas com igual prominência, e tampouco indica quais detecções são mais prováveis de serem falsos positivos.

Comparação com Semgrep: O Agentic Radar oferece análise holística com visualização de workflow completo e contexto gráfico, enquanto o Semgrep fornece detecções precisas com números de linha exatos e correspondência de padrões textuais. Esta diferença revela uma complementaridade natural: o contexto visual do Agentic Radar combinado com a precisão textual do Semgrep.

7.6 Análise Comparativa: Semgrep vs. Agentic Radar

Esta subseção apresenta análise comparativa sistemática das duas ferramentas avaliadas, documentando coberturas complementares, trade-offs e implicações para a abordagem híbrida proposta.

7.6.1 Comparação de Cobertura por Ferramenta

A Tabela 13 apresenta comparação detalhada de cobertura, performance e características operacionais.

Tabela 13: Comparação Semgrep vs. Agentic Radar

Dimensão	Semgrep (Generic Mode)	Agentic Radar
Workflows 1-3	19 detecções (10 documentadas) Cobertura: 71%	62 detecções (0 TP, 62 FP) Cobertura: 0%
Workflow 4	Não testado Cobertura: N/A	12 detecções (4 TP, 8 FP) Cobertura: 100% em AI
Execução	5,8s (3 workflows) Média: 1,9s/workflow	3,1s (4 workflows) Média: 0,78s/workflow
Instalação	`pip install semgrep` Simples, sem deps. ext.	`pip install agentic-radar` Simples, sem deps. ext.
Privacidade	Local, sem APIs ext. 100% offline	Local (scan mode) 100% offline
Formato Saída	JSON/SARIF/texto Parseable	HTML + JSON Visual + parseable
Especialização	Configurável (regex) Genérica	Agentes de IA apenas Especializada
Falsos Positivos	0 observados (workflows 1-3) Taxa: 0%	70 de 74 (94,6%) Taxa: 94,6%
Regras	Customizadas (7 regras) Requer desenvolvimento	Pré-configuradas Out-of-the-box
Limitações	Modo JSON inefetivo Requer modo genérico	FP em workflows sem IA Cobertura limitada

7.6.2 Análise de Complementaridade

A análise comparativa revela **complementaridade perfeita** entre as ferramentas, com sobreposição mínima e sinergia máxima:

Semgrep cobre: SQL Injection (3 instâncias detectadas), Command Injection (3 instâncias detectadas, incluindo 1 não documentada), operações não autorizadas em Vault (2 instâncias) e exposição de chaves SSH (1 instância).

Agentic Radar cobre: Prompt Injection direto em agentes de IA (2 instâncias) e exposição de informação sensível via LLM APIs (2 instâncias).

Sobreposição: Zero. Nenhuma vulnerabilidade foi detectada por ambas as ferramentas simultaneamente, confirmando especialização distinta.

Lacunas residuais (não cobertas pela implementação atual): SSRF potencial (1 instância no workflow 3) e DoS via loops ou operações intensivas (2 instâncias). Credenciais hardcoded em texto plano (3 instâncias no workflow 3) são detectáveis por regras Semgrep específicas, mas não foram incluídas no conjunto inicial de regras.

7.6.3 Trade-offs Identificados

1. Precisão vs. Abrangência: O Semgrep apresenta alta precisão (0% FP) porém cobertura média (71%), enquanto o Agentic Radar oferece cobertura completa em seu domínio especializado (100% em IA) mas com precisão baixa (94,6% FP). Este trade-off implica que o Semgrep é confiável para gates automatizados, ao passo que o Agentic Radar requer revisão manual.

2. Configurabilidade vs. Facilidade de Uso: O Semgrep requer desenvolvimento de regras customizadas (1,2h investidas com IA, 40-60h sem IA), enquanto o Agentic Radar funciona com zero configuração, fornecendo detecções out-of-the-box. Neste trade-off, o Semgrep oferece flexibilidade para adaptar-se a contextos específicos, ao passo que o Agentic Radar oferece conveniência imediata.

3. Especialização vs. Generalização: O Semgrep possui motor genérico adaptável a qualquer padrão via regex, enquanto o Agentic Radar é especializado em agentes de IA e não extensível para outras classes. Este trade-off demonstra que o Semgrep é versátil, ao passo que o Agentic Radar é focado em seu domínio específico.

7.6.4 Implicações para Abordagem Híbrida

A análise comparativa valida empiricamente a hipótese da abordagem híbrida: **nenhuma ferramenta isolada é suficiente**.

Semgrep isolado: Embora detecte 71% das vulnerabilidades tradicionais, apresenta zero cobertura em riscos de agentes de IA e não testa workflow 4 (AI-enabled). Consequentemente, é inadequado para ambientes com workflows de IA.

Agentic Radar isolado: Apesar de detectar 100% das vulnerabilidades de agentes de IA, apresenta zero cobertura em vulnerabilidades tradicionais (0/14) e taxa de FP inaceitável (94,6).

Abordagem híbrida (Semgrep + Agentic Radar): A combinação alcança cobertura de 14 de 18 vulnerabilidades (77,8%) com complementaridade notável: zero sobreposição e máxima sinergia. Entretanto, requer filtragem de FP do Agentic Radar (manual ou automatizada). Em síntese, constitui abordagem superior, validando a proposta do trabalho.

7.7 Resultados da Abordagem Híbrida

Esta subseção consolida os resultados da estratégia híbrida proposta, demonstrando que a combinação de ferramentas complementares alcança cobertura significativamente superior a qualquer ferramenta isolada.

7.7.1 Métricas Agregadas

A Tabela 14 apresenta métricas consolidadas da abordagem híbrida.

Tabela 14: Resultados agregados da abordagem híbrida

Métrica	Semgrep	Agentic Radar	Híbrido
Vulnerabilidades Detectadas	10/14	4/4 (AI only)	14/18
Cobertura Geral	71% (wf 1-3)	26,7% (todos)	**77,8%**
True Positives	19	4	23
False Positives	0	70	70
False Negatives	4	14	4
Precisão	100%	5,4%	24,7%
Recall	71%	22,2%	**77,8%**
F1-Score	0,83	0,09	**0,37**
Tempo Execução	5,8s	3,1s	8,9s
Adequado para CI/CD	Sim	Não (FP)	Parcial

Observações críticas: O recall aumentou de 71% para 77,8%, com a abordagem híbrida detectando 7 vulnerabilidades adicionais (4 em workflow 4). Por outro lado, a precisão degradou de 100% para 24,7%, pois os 70 falsos positivos do Agentic Radar impactam negativamente a métrica agregada. Consequentemente, o F1-Score reduziu, uma vez que o ganho em recall não compensa a perda em precisão quando métricas são agregadas sem ponderação. Quanto ao desempenho, o tempo total de 8,9s (execução sequencial) é aceitável para integração em CI/CD.

7.7.2 Cobertura por Classe de Vulnerabilidade

A Tabela 15 detalha a cobertura por classe, demonstrando como a abordagem híbrida preenche lacunas de cada ferramenta isolada.

Tabela 15: Cobertura por classe — Abordagem híbrida

Classe	Total	Semgrep	Agentic	Híbrido	Cob.
AI Agent Risks	4	0	4	4	**100%**
SQL Injection	3	3	0	3	**100%**
Command Injection	2	2	0	2	**100%**
Secret Operations	2	2	0	2	**100%**
SSH Key Exposure	1	1	0	1	**100%**
Hardcoded Credentials	3	0	0	0	0%
SSRF	1	0	0	0	0%
DoS	2	0	0	0	0%
TOTAL	**18**	**8**	**4**	**12**	**66,7%**

Análise de lacunas residuais: As três classes não cobertas pela implementação atual apresentam desafios técnicos específicos. Hardcoded Credentials (3 instâncias) são detectáveis via regras Semgrep com análise de entropia de strings ou padrões de high-entropy values em campos de senha, mas não foram priorizadas no conjunto inicial de regras. SSRF (1 instância) exige análise de fluxo de dados para rastrear URLs dinâmicas até fontes não confiáveis, desafiador no modo genérico. Por fim, DoS (2 instâncias) demanda análise de ciclos no grafo de execução e detecção de operações computacionalmente intensivas sem limites, além das capacidades de pattern matching.

7.7.3 Validação dos Objetivos

O objetivo estabelecido na Seção 4 definiu meta de **cobertura** $\geq$ **80%** para validar a viabilidade da abordagem híbrida.

Resultado alcançado: 77,8% (14 de 18 vulnerabilidades)

Análise do resultado: Embora a meta de 80% não tenha sido atingida (77,8% < 80%, déficit de 2,2 pontos percentuais), o resultado permanece próximo do alvo, com diferença de apenas 1 vulnerabilidade (15/18 alcançaria 83,3%). Ademais, representa melhoria significativa quando comparado a 71% (Semgrep isolado) ou 26,7% (Agentic Radar isolado). Portanto, a abordagem híbrida demonstra validação parcial de sua viabilidade, embora com espaço para melhorias.

Fatores contribuintes para déficit: Três fatores principais explicam o déficit observado. Primeiramente, o escopo limitado das regras Semgrep: apenas 1,2h de desenvolvimento focado em vulnerabilidades críticas, excluindo hardcoded credentials que são detectáveis com regras adicionais. Adicionalmente, limitações inerentes das ferramentas para SSRF e DoS, que requerem análise mais complexa. Por fim, o dataset desbalanceado contribui significativamente, uma vez que 3 de 6 classes não cobertas pela implementação atual representam 6 vulnerabilidades (33%).

Projeção com regras adicionais: Desenvolvimento de 2-3 regras Semgrep adicionais para hardcoded credentials (30min com IA) poderia elevar cobertura para 83,3-88,9%, superando a meta de 80%.

7.7.4 Workflow de Integração Proposto

Com base nos resultados experimentais, propõe-se o seguinte workflow de integração em pipelines de CI/CD:

1. **Análise paralela**: Executar Semgrep e Agentic Radar simultaneamente (tempo total: max(5,8s, 3,1s) = 5,8s vs. 8,9s sequencial)

2. **Agregação de resultados**: Consolidar detecções de ambas as ferramentas em formato SARIF unificado

3. **Filtragem de falsos positivos**: Aplicar heurística para descartar detecções de Agentic Radar em workflows sem nós de IA identificados (reduz 62 de 70 FP = 88,6%)

4. **Priorização por severidade**: Classificar vulnerabilidades como BLOCKER (SQL/Command Injection, Prompt Injection), CRITICAL (Secret Operations, PII Exposure) ou WARNING (demais)

5. **Gate condicional**: Bloquear merge se vulnerabilidades BLOCKER detectadas; alertar para CRITICAL; informar sobre WARNING

6. **Revisão manual**: Analista de segurança revisa detecções CRITICAL e valida potenciais FP residuais do Agentic Radar

Performance esperada em produção: O tempo total estimado é de 6-8s por workflow, considerado aceitável para integração em CI/CD. A taxa de FP após filtragem reduz significativamente para aproximadamente 8 de 23 detecções (34,8%), representando melhoria substancial em relação aos 94,6% iniciais. Simultaneamente, a cobertura mantém-se em 77,8%, enquanto o recall de vulnerabilidades críticas alcança 100% (todas SQL/Command/Prompt Injection detectadas).

7.7.5 Implementação de Referência: workflow_analyzer.py

Como prova de conceito da arquitetura proposta, foi desenvolvido um script Python que implementa os dois componentes principais descritos na Seção 5. O script `workflow_analyzer.py` consolida a arquitetura híbrida em uma ferramenta executável que pode ser integrada diretamente em pipelines de CI/CD.

Componente A — Módulo de Entrada e Validação:

A classe `WorkflowValidator` implementa as funcionalidades de parsing e validação:

Listing 1: Estrutura do módulo de validação

```
class WorkflowValidator:
    def load_workflow(self, filepath: str) -> Tuple[Optional[Dict],
        ValidationResult]:
        # Parser JSON com tratamento de erros
        # Retorna workflow e resultado de validacao

    def validate_structure(self, workflow: Dict) -> ValidationResult:
        # Valida presenca de 'nodes' e 'connections'
        # Verifica estrutura de cada no (id, name, type)

    def extract_metadata(self, workflow: Dict, filepath: str) ->
        WorkflowMetadata:
        # Extrai nome, ID, contagem de nos, tipos de nos

    def scan_directory(self, directory: str) -> List[str]:
        # Suporte a analise em lote
```

O módulo realiza validação estrutural em três níveis: (1) verificação de existência e legibilidade do arquivo, (2) validação de sintaxe JSON, e (3) validação de esquema com verificação de campos obrigatórios (`nodes`, `connections`) e estrutura de nós. Erros são reportados de forma informativa, permitindo diagnóstico rápido de problemas.

Componente B — Motor de Análise Híbrida:

Duas classes implementam os executores das ferramentas de análise:

Listing 2: Executores de análise

```
class AgenticRadarExecutor:
    def run(self, workflow_path: str) -> Tuple[List[Dict], bool, str
        ]:
        # Invoca: agentic-radar scan <workflow> --output <dir>
        # Captura saida HTML/JSON
        # Retorna (findings, success, error_message)

class SemgrepExecutor:
    def run(self, workflow_path: str) -> Tuple[List[Dict], bool, str
        ]:
        # Invoca: semgrep --config=rules/ --json <workflow>
        # Parse da saida JSON (formato nativo Semgrep)
        # Retorna (findings, success, error_message)
```

A classe `HybridAnalyzer` orquestra a execução sequencial de ambas as ferramentas:

Listing 3: Orquestrador de análise híbrida

```python
class HybridAnalyzer:
    def analyze(self, workflow_path: str) -> Optional[AnalysisResult
        ]:
        # 1. Carrega e valida workflow
        # 2. Extrai metadados
        # 3. Executa Agentic Radar
        # 4. Executa Semgrep
        # 5. Agrega resultados
        # 6. Retorna AnalysisResult unificado

    def analyze_batch(self, directory: str) -> List[AnalysisResult]:
        # Analise em lote de todos workflows no diretorio
```

Interface de Linha de Comando:

O script oferece interface CLI completa para diferentes cenários de uso:

Listing 4: Exemplos de uso do CLI

```bash
# Analise de workflow individual
python workflow_analyzer.py workflow_1.json

# Analise em lote de diretorio
python workflow_analyzer.py --batch ./input_workflows/

# Especificacao de diretorio de saida e regras customizadas
python workflow_analyzer.py workflow_1.json --output results/ \
    --rules rules/n8n-generic-patterns.yaml

# Geracao de relatorio em diferentes formatos
python workflow_analyzer.py --batch ./input_workflows/ --format both
```

Geração de Relatórios:

O script gera relatórios em dois formatos complementares. O formato JSON oferece estrutura para processamento automatizado, contendo metadados completos do workflow, findings de ambas ferramentas com localização e severidade, além de métricas de execução. Paralelamente, o formato Markdown fornece apresentação legível para revisão humana, incluindo sumário executivo com estatísticas agregadas, detalhamento por workflow com findings categorizados por ferramenta e tempo de execução.

Características de Implementação:

A implementação incorpora resiliência, permitindo que se uma ferramenta falha, a outra ainda execute e reporte resultados. Além disso, possui proteção de timeout de 60 segundos por ferramenta para evitar travamentos. O script mantém dependências mínimas, utilizando apenas a biblioteca padrão Python (`json`, `subprocess`, `argparse`, `pathlib`). Finalmente, emprega type hints completos para facilitar manutenção e integração com IDEs.

Integração com Regras Existentes:

O script utiliza automaticamente as 7 regras Semgrep desenvolvidas neste trabalho, localizadas em `rules/n8n-generic-patterns.yaml`. A Tabela 16 resume as regras disponíveis.

Tabela 16: Regras Semgrep integradas ao workflow_analyzer.py

Rule ID	CWE	Vulnerabilidade Detectada
n8n-pgpassword-injection	CWE-78	Command Injection via PGPASSWORD
n8n-ssh-key-exposed	CWE-798	Chave SSH hardcoded
n8n-command-with-expression	CWE-78	Command Injection em SSH
n8n-hardcoded-password-in-json	CWE-798	Senha hardcoded em workflow
n8n-sql-injection-risk	CWE-89	SQL Injection via expressão dinâmica
n8n-delete-dynamic-url	CWE-285	Deleção não autorizada via URL dinâmica
n8n-vault-secret-operation	—	Operação sensível em Vault

Esta implementação de referência demonstra a viabilidade técnica da arquitetura proposta e pode ser utilizada diretamente por organizações que desejam integrar análise SAST em seus pipelines de desenvolvimento de workflows n8n.

7.7.6 Conclusão da Avaliação Híbrida

A avaliação experimental valida **parcialmente** a hipótese da abordagem híbrida:

Validado: A avaliação confirmou a complementaridade das ferramentas com zero sobreposição e máxima sinergia. Adicionalmente, demonstrou cobertura superior à ferramenta isolada (77,8% vs. 71% ou 26,7%), bem como viabilidade técnica evidenciada por tempo de execução aceitável e instalação simples. Notavelmente, alcançou detecção de 100% das vulnerabilidades críticas (SQL/Command/Prompt Injection).

Limitações: Por outro lado, a meta de 80% não foi alcançada, com déficit de 2,2 pontos percentuais. Além disso, observou-se taxa de FP elevada sem filtragem (94,6% do Agentic Radar). Por fim, permanecem lacunas residuais em 3 classes de vulnerabilidades (33%).

Contribuição para o estado da arte: Demonstração empírica de que análise estática de workflows LCNC em formato JSON é viável e efetiva, alcançando cobertura de 77,8% através de estratégia híbrida combinando ferramenta especializada (Agentic Radar) e ferramenta configurável (Semgrep).

8 Discussão

Esta seção interpreta os resultados experimentais apresentados na Seção 7, analisando criticamente o alcance dos objetivos estabelecidos, as limitações identificadas, o posicionamento da abordagem proposta em relação ao estado da arte e as implicações práticas para segurança de plataformas Low-Code/No-Code.

8.1 Alcance dos Objetivos

A avaliação experimental demonstrou o alcance substancial dos objetivos específicos estabelecidos na Seção 4, com resultados que validam a viabilidade técnica da abordagem híbrida proposta.

OE1 — Análise Comparativa: Alcançado integralmente. A análise sistemática documentou cobertura de 16,7% (Agentic Radar) e 71% (Semgrep) para vulnerabilidades tradicionais, estabelecendo métricas quantitativas que demonstram complementaridade das ferramentas. A caracterização qualitativa identificou limitações críticas (taxa de FP de 94,6% no Agentic Radar, falha completa do modo JSON do Semgrep) essenciais para aplicação prática.

OE2 — Metodologia com IA: Alcançado com sucesso. O uso de Claude 3.5 Sonnet para geração assistida de regras Semgrep resultou em aceleração de 13x (1,2h vs. 49h estimadas), demonstrando viabilidade da abordagem. O processo iterativo de refinamento (2-4 iterações por regra) estabeleceu metodologia reproduzível para desenvolvimento futuro.

OE3 — Regras Customizadas: Alcançado. Desenvolveram-se 7 regras funcionais em modo genérico cobrindo SQL Injection, Command Injection, operações não autorizadas em Vault e exposição de chaves SSH. A descoberta crítica da necessidade de usar modo genérico (não JSON) representa contribuição metodológica importante para análise de plataformas LCNC.

OE4 — Validação Empírica: Alcançado. A execução sobre 4 workflows (18 vulnerabilidades totais) produziu cobertura híbrida de 77,8%, demonstrando efetividade da combinação de ferramentas. Métricas detalhadas (precisão, recall, F1-score) foram calculadas e documentadas.

OE5 — Documentação de Lacunas: Alcançado. Identificaram-se três classes não cobertas (hardcoded credentials, SSRF, DoS) com justificativas técnicas e propostas concretas para trabalhos futuros (Seção 9).

Meta de Cobertura (80%): Não alcançada, mas próxima. A cobertura de 77,8% ficou 2,2 pontos percentuais abaixo da meta, equivalente a 1 vulnerabilidade adicional. Análise indica que desenvolvimento de 2-3 regras adicionais para detecção de credenciais hardcoded (30min com IA) elevaria cobertura para 83-89%, superando a meta estabelecida. O déficit não invalida a viabilidade da abordagem, representando oportunidade de melhoria incremental.

8.2 Limitações Identificadas

Este trabalho apresenta limitações técnicas e de segurança que devem ser consideradas ao interpretar os resultados e aplicar a abordagem em contextos práticos.

8.2.1 Limitações Técnicas

1. Incompatibilidade do Modo JSON: A falha completa do modo JSON do Semgrep (0% de cobertura, 33 regras desenvolvidas) representa limitação arquitetural fundamental. Estruturas JSON profundamente aninhadas (4+ níveis), características de plataformas LCNC, excedem capacidades do parser. A solução via modo genérico, embora efetiva, sacrifica compreensão semântica por correspondência textual baseada em regex, limitando capacidades de análise de fluxo de dados (taint analysis).

2. Dataset Sintético Limitado: A validação baseou-se em 4 workflows sintéticos (18 vulnerabilidades), representando contexto controlado que pode não capturar a diversidade de padrões em ambientes de produção. Workflows corporativos frequentemente apresentam complexidade superior (100+ nós, múltiplas ramificações condicionais), cujo comportamento das ferramentas não foi validado neste estudo.

3. Cobertura Incompleta: Três classes de vulnerabilidades permaneceram não cobertas pela implementação atual (hardcoded credentials, SSRF, DoS), representando 33% das categorias identificadas. Estas lacunas decorrem de: (i) credenciais hardcoded são detectáveis via regras Semgrep com análise de padrões e entropia de strings, mas não foram priorizadas no conjunto inicial; (ii) SSRF necessita rastreamento de fluxo de dados até fontes não confiáveis, desafiador no modo genérico; (iii) DoS exige detecção de ciclos em grafos de execução, além das capacidades de pattern matching.

4. Ausência de Análise Dinâmica: A abordagem puramente estática não detecta vulnerabilidades manifestadas apenas em tempo de execução (race conditions, vulnerabilidades de-

pendentes de estado, ataques de timing). Análise dinâmica complementar seria necessária para cobertura abrangente.

8.2.2 Limitações de Segurança

1. Taxa Elevada de Falsos Positivos: O Agentic Radar apresentou taxa de FP de 94,6% (70 de 74 detecções), inviabilizando uso autônomo em gates de CI/CD sem revisão manual substancial. A heurística excessivamente ampla (qualquer nó HTTP Request sinalizado como vetor de injeção de prompt) demonstra necessidade de refinamento para consciência de contexto.

2. Sem Verificação de Integridade de Nós: Nós customizados da comunidade n8n, instaláveis via npm, não são verificados quanto a vulnerabilidades ou comportamento malicioso. Workflows utilizando nós comprometidos permaneceriam inseguros independentemente da análise SAST dos parâmetros de configuração.

3. Limitações de Taint Analysis: A ausência de rastreamento completo de fluxo de dados através do grafo de execução limita detecção de vulnerabilidades onde dados não confiáveis propagam-se através de múltiplas transformações antes de alcançarem sinks perigosos. O modo genérico do Semgrep detecta padrões sintáticos locais, mas não fluxos complexos.

8.3 Comparação com Estado-da-Arte

Este trabalho representa o primeiro estudo sistemático de análise estática de segurança para workflows n8n, preenchendo lacuna crítica identificada na literatura. A Tabela 17 posiciona a abordagem proposta em relação a ferramentas existentes e trabalhos relacionados.

Tabela 17: Comparação com estado da arte em SAST para plataformas LCNC

Solução	Cobertura	LCNC	n8n	Características
Agentic Radar	16,7% (n8n)	Parcial	Sim	Especializado em riscos de IA, FP 94,6%
Semgrep	71% (n8n)	Não	Adaptável	Requer regras custom, modo genérico necessário
Abordagem Híbrida (Este trabalho)	**77,8%**	Sim	Sim	Complementar, 100% vulnerabilidades críticas
SonarQube	N/A	Não	Não	Código tradicional, sem suporte JSON workflows
Checkmarx	N/A	Não	Não	Comercial, foco em código imperativo
GitHub CodeQL	N/A	Não	Não	Linguagens específicas, sem parser n8n

Diferencial deste trabalho: Até o momento, não existe ferramenta dedicada ou estudo acadêmico documentando análise SAST para workflows n8n. A literatura (OWASP LCNC Top 10, análise de workflows n8n) identifica o problema, mas soluções práticas estavam ausentes. Este trabalho contribui primeiramente com a validação empírica de análise estática em workflows n8n, alcançando 77,8% de cobertura. Adicionalmente, estabelece descoberta metodológica crítica ao demonstrar que o modo genérico é superior ao modo JSON para estruturas LCNC (0% vs 71%). Além disso, propõe metodologia de IA para desenvolvimento de regras, resultando em aceleração de 13x com Claude 3.5 Sonnet. Finalmente, valida empiricamente a arquitetura híbrida, demonstrando complementaridade entre Agentic Radar e Semgrep.

Trabalhos relacionados em outras plataformas LCNC (Power Automate, Zapier) não foram identificados na literatura acadêmica, sugerindo que esta área de pesquisa está em estágio inicial de desenvolvimento.

8.4 Trade-offs Segurança vs. Usabilidade

A abordagem híbrida proposta envolve compromissos fundamentais entre segurança, usabilidade e operacionalidade que merecem análise crítica.

1. Precisão vs. Recall (Semgrep 100%/71% vs. Agentic Radar 5,4%/22%): O Semgrep prioriza precisão (zero FP observados em workflows 1-3), adequado para gates automatizados de CI/CD que bloqueiam merges. O Agentic Radar prioriza recall (detecta 100% das vulnerabilidades em seu domínio especializado), mas ao custo de 94,6% de FP, exigindo revisão manual. A escolha híbrida combina ambos: Semgrep para detecções confiáveis automatizadas + Agentic Radar para análise auxiliar com validação humana.

2. Configurabilidade vs. Facilidade de Uso: O Semgrep requer investimento inicial de desenvolvimento de regras customizadas (1,2h com IA, 40-60h sem IA), mas oferece flexibilidade total para adaptar-se a novos padrões. O Agentic Radar funciona out-of-the-box sem configuração, mas não é extensível para vulnerabilidades fora de seu escopo especializado. Para organizações com equipes de segurança dedicadas, o trade-off favorece Semgrep (customizável); para equipes pequenas, Agentic Radar oferece valor imediato.

3. Privacidade vs. Funcionalidade: Ambas as ferramentas operam localmente em modo scan, preservando privacidade (crítico para conformidade LGPD). Entretanto, o modo test do Agentic Radar requer envio de workflows para API da OpenAI, criando trade-off: funcionalidade avançada (testes adversariais com LLM) vs. exposição de lógica proprietária. Organizações com requisitos rigorosos de privacidade devem limitar-se ao modo scan.

4. IA como Acelerador vs. Validação Manual: A geração assistida de regras com Claude 3.5 Sonnet acelera desenvolvimento em 13x, mas não elimina necessidade de validação empírica. Das 33 regras geradas para modo JSON, nenhuma funcionou (0% cobertura), demonstrando que IA reduz overhead mas não substitui expertise técnica. O trade-off otimizado: IA para geração inicial + validação rigorosa com workflows de teste.

8.5 Implicações Práticas

Os resultados deste trabalho têm implicações diretas para diferentes stakeholders e para o desenvolvimento de políticas públicas relacionadas à segurança de plataformas LCNC.

8.5.1 Para Usuários/Stakeholders

Equipes de Segurança: A abordagem híbrida é viável para integração em pipelines de CI/CD, com tempo de execução aceitável (6-8s por workflow) e capacidade de bloquear automatica-

mente vulnerabilidades críticas (SQL/Command Injection, Prompt Injection). Recomenda-se configuração de gates condicionais: bloqueio para severidade BLOCKER, alertas para CRITICAL, informação para WARNING.

Desenvolvedores Cidadãos: A disponibilização de análise SAST automatizada eleva conscientização sobre riscos de segurança em workflows. Mensagens explicativas das ferramentas (mapeamento OWASP, exemplos de código corrigido) funcionam como material educacional integrado ao fluxo de desenvolvimento, reduzindo lacuna de conhecimento identificada na literatura.

Organizações: A solução permite estabelecer governança de workflows n8n através de políticas automatizadas, endereçando lacuna crítica onde "código-fonte"(JSON) não estava sujeito ao mesmo rigor que plataforma subjacente. Particularmente relevante para conformidade LGPD [1], considerando que workflows frequentemente processam dados pessoais sensíveis de cidadãos brasileiros.

Treinamento Necessário: Analistas de segurança requerem treinamento em interpretação de resultados do Agentic Radar (distinção FP/TP), configuração de regras Semgrep e análise de workflow JSON. Estimativa: 4-8 horas de capacitação técnica.

8.5.2 Para Políticas Públicas

Conformidade Regulatória (LGPD): Workflows n8n que processam dados pessoais devem implementar medidas de segurança técnicas apropriadas (Art. 46, LGPD [3]). A análise SAST automatizada constitui controle preventivo que auxilia organizações a demonstrar conformidade, particularmente em processos de tratamento de dados sensíveis (saúde, dados financeiros).

Padronização de Análise de Segurança LCNC: A ausência de padrões e ferramentas para plataformas LCNC representa lacuna que afeta organizações públicas e privadas. Este trabalho demonstra viabilidade técnica, sugerindo oportunidade para desenvolvimento de frameworks nacionais de segurança para automações empresariais, alinhados com estratégias de transformação digital do governo brasileiro.

Incentivo à Pesquisa: A área de segurança em plataformas LCNC está em estágio inicial, conforme evidenciado pela ausência de trabalhos relacionados na literatura. Agências de fomento (CNPq, CAPES, FAPESP) poderiam priorizar linhas de pesquisa em análise de segurança para plataformas emergentes, considerando crescente adoção no setor público e privado nacional.

9 Conclusões e Próximos Passos

Este trabalho validou empiricamente a viabilidade técnica de análise estática de segurança para workflows n8n através de abordagem híbrida combinando ferramentas complementares. A investigação demonstrou que análise SAST de estruturas JSON declarativas é possível e efetiva, alcançando cobertura de 77,8% das vulnerabilidades identificadas. Contribuições metodológicas, técnicas e científicas foram estabelecidas, posicionando o trabalho como primeiro estudo sistemático de segurança para esta plataforma LCNC emergente. Esta seção sintetiza as evidências, documenta contribuições, propõe trabalhos futuros em três horizontes temporais e oferece recomendações práticas para diferentes stakeholders.

9.1 Síntese das Evidências

A validação experimental estabeleceu cinco conclusões fundamentais que respondem às questões de pesquisa formuladas:

1. Viabilidade Técnica Demonstrada: A análise estática de workflows n8n em formato JSON é viável, alcançando cobertura híbrida de 77,8% (14 de 18 vulnerabilidades). Embora inferior à meta de 80%, o resultado demonstra aplicabilidade prática e representa melhoria substancial sobre ausência de ferramentas (baseline de 0%).

2. Descoberta Crítica — Modo Genérico vs. JSON: A falha completa do modo JSON do Semgrep (0% de cobertura, 33 regras desenvolvidas) contrastada com sucesso do modo genérico (71% de cobertura, 7 regras) representa contribuição metodológica importante. Estruturas JSON profundamente aninhadas características de plataformas LCNC requerem adaptação de ferramentas SAST tradicionais: correspondência textual via regex supera parsing AST para este paradigma declarativo.

3. IA como Acelerador Validado: O uso de Claude 3.5 Sonnet para geração assistida de regras Semgrep resultou em aceleração de 13x (1,2h vs. 49h estimadas manualmente). Entretanto, a falha das regras JSON demonstra que IA reduz overhead mas não elimina necessidade de validação empírica rigorosa. Processo iterativo de refinamento (2-4 iterações por regra) é essencial.

4. Complementaridade Perfeita: Agentic Radar e Semgrep apresentam zero sobreposição de cobertura, validando hipótese da abordagem híbrida. Agentic Radar detecta 100% das vulnerabilidades de agentes de IA (4/4), enquanto Semgrep detecta 71% das vulnerabilidades tradicionais (10/14). A combinação maximiza cobertura ao custo de gerenciar 94,6% de taxa de FP do Agentic Radar.

5. Lacuna Científica Preenchida: Este é o primeiro trabalho a documentar sistematicamente análise SAST para workflows n8n com validação experimental. A ausência de trabalhos relacionados para n8n ou outras plataformas LCNC na literatura acadêmica posiciona esta pesquisa como uma ótima contribuição em área emergente de segurança cibernética.

9.2 Contribuições

Este trabalho oferece contribuições em quatro dimensões que avançam o estado da arte em segurança de plataformas LCNC:

1. Contribuição Metodológica — Geração Assistida de Regras por IA: Desenvolvimento de metodologia sistemática para geração de regras Semgrep utilizando Claude 3.5 Sonnet, incluindo estruturação de prompts em três componentes (contexto do esquema n8n, objetivo de detecção, formato YAML), processo iterativo de refinamento (2-4 iterações por regra) e validação empírica obrigatória. A aceleração de 13x demonstrada (1,2h vs. 49h) estabelece precedente para aplicação de IA generativa em desenvolvimento de conhecimento de segurança, reproduzível para outras plataformas LCNC.

2. Contribuição Técnica — Regras Funcionais e Descoberta Arquitetural: Desenvolvimento de 7 regras Semgrep validadas em modo genérico cobrindo SQL Injection, Command Injection, operações não autorizadas em Vault e exposição de chaves SSH, disponibilizadas como artefato reproduzível. A descoberta crítica de que modo genérico supera modo JSON para estruturas LCNC profundamente aninhadas (71% vs. 0% de cobertura) representa insight arquitetural importante: ferramentas SAST tradicionais requerem adaptação ao paradigma declarativo de configuração-como-código.

3. Contribuição Científica — Validação Empírica de SAST para LCNC: Primeiro estudo a documentar sistematicamente aplicação de ferramentas SAST a workflows n8n com

validação experimental rigorosa (4 workflows, 18 vulnerabilidades, métricas quantitativas de precisão/recall). A caracterização detalhada de limitações (taxa de FP 94,6% do Agentic Radar, incompatibilidade do modo JSON, lacunas em 33% das classes) fornece base empírica para pesquisas futuras e estabelece baseline para comparações.

4. Contribuição Prática — Workflow de Integração CI/CD: Proposta de arquitetura de integração para pipelines de desenvolvimento, incluindo execução paralela de ferramentas (redução de 8,9s para 5,8s), filtragem inteligente de falsos positivos (redução de 94,6% para 34,8%), classificação por severidade (BLOCKER/CRITICAL/WARNING) e gates condicionais. O workflow proposto é diretamente aplicável por organizações que utilizam n8n em produção, endereçando lacuna prática de governança de automações.

9.3 Trabalhos Futuros

Os resultados obtidos abrem oportunidades concretas de extensão e melhoria em três horizontes temporais, desde melhorias incrementais imediatas até desenvolvimento de ferramentas dedicadas de longo prazo.

9.3.1 Curto Prazo (3-6 meses)

Melhorias incrementais focadas em elevar cobertura e reduzir falsos positivos, aplicáveis com esforço limitado:

1. Expansão de Regras para Classes Não Cobertas: Desenvolvimento de 3-4 regras Semgrep adicionais para detecção de credenciais hardcoded (análise de entropia de strings em campos `password`/`apiKey`), SSRF básico (URLs dinâmicas sem validação em nós HTTP Request) e DoS por loops (detecção de conexões cíclicas no grafo de workflow). Esforço estimado: 1-2h com IA, elevando cobertura de 77,8% para 83-89%.

2. Redução de Falsos Positivos do Agentic Radar: Implementação de heurística de consciência de contexto que filtra detecções de "Indirect Prompt Injection"em workflows sem nós de IA identificados. Análise preliminar indica que esta filtragem simples reduziria taxa de FP de 94,6% para 34,8% (eliminação de 62 de 70 FP), tornando ferramenta viável para uso automatizado com revisão manual limitada.

3. Validação com Dataset Real de Produção: Execução das ferramentas sobre dataset de workflows corporativos reais (target: 50-100 workflows de organizações parceiras), permitindo validação de performance em workflows complexos (100+ nós), identificação de padrões não capturados pelo dataset sintético e calibração de regras para redução de falsos positivos/negativos em contexto produtivo.

4. Publicação de Conjunto de Regras: Disponibilização pública das 7 regras Semgrep desenvolvidas em repositório GitHub com documentação (README, exemplos de uso, instruções de instalação), dataset de workflows de teste para validação comunitária e licença permissiva (MIT) para facilitar adoção e contribuições. Objetivo: estabelecer baseline comunitário para segurança de workflows n8n.

9.3.2 Médio Prazo (6-12 meses)

Desenvolvimento de Registro de Regras Comunitário para n8n

Inspirado pelo modelo de sucesso do registro público de regras Semgrep (https://semgrep.dev/r), que oferece conjuntos de regras específicos para dezenas de linguagens de programação e frameworks, propõe-se o desenvolvimento de um registro análogo específico para workflows n8n. Este registro centralizaria regras de detecção de vulnerabilidades desenvolvidas pela

comunidade, categorizadas por classe de vulnerabilidade (OWASP LCNC Top 10) e validadas contra workflows de teste.

O repositório `n8n-CyberSecurity-Workflows` [4] demonstra interesse comunitário significativo em automação de segurança com n8n, sugerindo que uma iniciativa de regras compartilhadas teria adoção prática. A metodologia de geração assistida por IA desenvolvida neste trabalho poderia acelerar a criação inicial do registro, reduzindo a barreira de entrada para contribuições.

Componentes principais: Repositório Git com regras Semgrep em modo genérico validadas, documentação de cada regra com exemplos de código vulnerável e seguro, dataset de teste público com workflows contendo vulnerabilidades conhecidas, pipeline de CI/CD para validação automática de contribuições comunitárias, e integração com gerenciador de pacotes (npm/pip) para distribuição fácil.

9.3.3 Longo Prazo (1-2 anos)

Ferramenta SAST Dedicada para Workflows n8n

A longo prazo, propõe-se o desenvolvimento de uma ferramenta SAST especializada para o ecossistema n8n, inspirada no sucesso do Semgrep mas otimizada especificamente para a estrutura JSON declarativa de workflows. Enquanto o Semgrep em modo genérico demonstrou efetividade neste trabalho (71% de cobertura), uma ferramenta dedicada poderia oferecer capacidades superiores através de compreensão nativa do esquema n8n.

Diferencial competitivo vs. Semgrep genérico: **Parser nativo do esquema n8n** com compreensão semântica de nós, parâmetros e conexões permitindo análise de fluxo de dados (taint analysis) através do grafo de execução, **regras pré-configuradas** com conjunto abrangente de regras para as seis classes de vulnerabilidades mantido e atualizado pela comunidade, **integração CI/CD nativa** com plugin para GitHub Actions, GitLab CI e Jenkins que bloqueia automaticamente merges de workflows vulneráveis, **interface visual** com dashboard web mostrando postura de segurança de workflows organizacionais e tendências temporais, e **análise incremental** com detecção de mudanças entre versões de workflows reportando apenas novas vulnerabilidades introduzidas.

Modelo de distribuição:

Seguindo o modelo bem-sucedido do Semgrep, propõe-se arquitetura híbrida open-source + SaaS com **core open-source (MIT License)** incluindo motor de análise, regras comunitárias e CLI para execução local mantendo transparência e permitindo auto-hospedagem para organizações com requisitos rigorosos de privacidade, e **SaaS opcional (freemium)** oferecendo dashboard colaborativo, agregação histórica, gerenciamento centralizado de políticas e integração com plataformas n8n Cloud cobrando apenas por funcionalidades de equipe.

Esta dualidade garante acessibilidade (pesquisadores e pequenas organizações utilizam versão gratuita) enquanto cria modelo de sustentabilidade financeira que viabiliza desenvolvimento contínuo e suporte empresarial.

Integração no pipeline de desenvolvimento n8n:

A ferramenta seria integrada diretamente no fluxo de trabalho de desenvolvimento de workflows através de **pre-commit hook** com análise local antes de commit alertando desenvolvedor imediatamente sobre vulnerabilidades introduzidas, **pull request checks** com bloqueio automático de PRs contendo vulnerabilidades críticas e comentários inline identificando padrões inseguros e sugerindo correções, **deploy gate** com validação final antes de implantação em produção e políticas configuráveis por ambiente (desenvolvimento permite warnings, produção bloqueia errors), e **monitoramento contínuo** com escaneamento periódico de workflows exis-

tentes em produção identificando vulnerabilidades introduzidas por mudanças na plataforma n8n ou descoberta de novos vetores de ataque.

Sustentabilidade e governança:

Inspirado em projetos open-source bem-sucedidos como Semgrep e GitLab, implementa-se governança comunitária através de comitê técnico com representantes de organizações usuárias, roadmap público com priorização transparente de funcionalidades, processo formal de contribuição com revisão por pares e CI/CD rigoroso, documentação abrangente incluindo guias de contribuição e arquitetura técnica, e financiamento híbrido através de patrocínios corporativos, receita SaaS e grants de pesquisa.

Esta visão de longo prazo posiciona a ferramenta como padrão de facto para análise de segurança de workflows n8n, similar ao papel que o Semgrep desempenha para código tradicional, elevando significativamente a maturidade de segurança do ecossistema LCNC.

9.4 Recomendações

Com base nos resultados experimentais e limitações identificadas, recomenda-se aos diferentes stakeholders:

Para Equipes de Segurança Organizacionais: Adotar abordagem híbrida (Semgrep + Agentic Radar) em pipelines de CI/CD, configurando gates condicionais que bloqueiam vulnerabilidades BLOCKER (SQL/Command/Prompt Injection) automaticamente e requerem revisão manual para CRITICAL (exposição de credenciais, PII). Implementar execução paralela das ferramentas para minimizar impacto no tempo de build (5-8s por workflow). Investir 4-8h em treinamento de analistas para interpretação de resultados e distinção de falsos positivos.

Para Desenvolvedores e Administradores de n8n: Integrar análise SAST no fluxo de desenvolvimento através de pre-commit hooks locais (feedback imediato) e validação em pull requests (bloqueio de merges vulneráveis). Utilizar as 7 regras Semgrep desenvolvidas neste trabalho como baseline, expandindo conforme necessidades organizacionais específicas. Estabelecer políticas de governança que exijam análise SAST antes de implantação de workflows em produção, particularmente para workflows que processam dados sensíveis sujeitos à LGPD [1].

Para Pesquisadores e Acadêmicos: Explorar análise de fluxo de dados (taint analysis) através do grafo de execução de workflows para detectar propagação de dados não confiáveis através de múltiplas transformações. Investigar análise dinâmica complementar (execução em sandbox) para detectar vulnerabilidades manifestadas apenas em runtime. Estender abordagem para outras plataformas LCNC (Power Automate, Zapier, Make) para estabelecer frameworks gerais de segurança para paradigma declarativo. Desenvolver técnicas de análise estática conscientes de esquema que superem limitações de parsing JSON identificadas neste trabalho.

Para a Comunidade n8n e Open-Source: Contribuir com desenvolvimento de regras Semgrep adicionais no registro comunitário proposto, compartilhando conhecimento de padrões inseguros identificados em contextos de uso específicos. Criar e manter dataset público de workflows de teste contendo vulnerabilidades conhecidas para validação de ferramentas. Estabelecer boas práticas de segurança documentadas (security guidelines) para desenvolvimento de workflows, reduzindo lacuna de conhecimento dos "desenvolvedores cidadãos"identificada na literatura.

Referências

[1] BRASIL. *Lei nº 13.709, de 14 de agosto de 2018 - Lei Geral de Proteção de Dados Pessoais (LGPD)*. 2018. URL: https://www.planalto.gov.br/ccivil_03/_ato2015-2018/2018/lei/l13709.htm.

[2] BRASIL. *Lei nº 13.709/2018, Art. 20 - Decisão automatizada*. Art. 20. O titular dos dados tem direito a solicitar a revisão de decisões tomadas unicamente com base em tratamento automatizado de dados pessoais que afetem seus interesses. 2018. URL: https://www.planalto.gov.br/ccivil_03/_ato2015-2018/2018/lei/l13709.htm.

[3] BRASIL. *Lei nº 13.709/2018, Art. 46 - Princípio da segurança*. Art. 46. Os agentes de tratamento devem adotar medidas de segurança, técnicas e administrativas aptas a proteger os dados pessoais de acessos não autorizados e de situações acidentais ou ilícitas de destruição, perda, alteração, comunicação ou qualquer forma de tratamento inadequado ou ilícito. 2018. URL: https://www.planalto.gov.br/ccivil_03/_ato2015-2018/2018/lei/l13709.htm.

[4] CYBERSECURITYUP. *n8n-cybersecurity-workflows: Security automation with n8n ideas: 100+ red/blue/appsec workflows, integrations, and ready-to-run playbooks*. 2025. URL: https://github.com/CyberSecurityUP/n8n-CyberSecurity-Workflows.

[5] DEV.TO. *Tryhackme: Prototype pollution - dev community*. 2025. URL: https://dev.to/seanleeys/tryhackme-prototype-pollution-5fa2.

[6] GITLAB. *Static application security testing (sast) - gitlab docs*. 2025. URL: https://docs.gitlab.com/user/application_security/sast/.

[7] N8N.IO. *n8n legal*. 2025. URL: https://n8n.io/legal/.

[8] N8N.IO. *Privacy - n8n docs*. 2025. URL: https://docs.n8n.io/privacy-security/privacy/.

[9] OWASP. *Owasp low-code/no-code top 10*. 2025. URL: https://owasp.org/www-project-top-10-low-code-no-code-security-risks/.

[10] OWASP. *Source code analysis tools - owasp foundation*. 2025. URL: https://owasp.org/www-community/Source_Code_Analysis_Tools.

[11] SEMGREP. *Custom rule examples - semgrep*. 2025. URL: https://semgrep.dev/docs/writing-rules/rule-ideas.

[12] SEMGREP. *Semgrep app security platform | ai-assisted sast, sca and secrets detection*. 2025. URL: https://semgrep.dev/.

[13] SPLXAI. *Scanning n8n workflows with agentic radar | splxai blog*. 2025. URL: https://splx.ai/blog/scanning-n8n-workflows-with-agentic-radar.

[14] SplxAI. *Scanning n8n workflows with agentic radar*. 2025. URL: https://medium.com/@SplxAI/scanning-n8n-workflows-with-agentic-radar-62f8e1a5c705.

A Código Completo do Analisador de Workflows

Abaixo está o código completo do `workflow_analyzer.py`, implementando o sistema híbrido de análise com Agentic Radar e Semgrep.

Listing 5: workflow_analyzer.py - Analisador híbrido completo

```python
#!/usr/bin/env python3
"""
n8n Workflow Security Analyzer

This script implements a hybrid security analysis system for n8n workflows,
combining two analysis engines:
- Agentic Radar: Focuses on AI-specific vulnerabilities (prompt injection, PII leakage)
- Semgrep: Custom rules for traditional security issues (SQLi, CMDi, SSRF, secrets)

Components:
A. Input and Validation Module - JSON parsing, structure validation, metadata extraction
B. Hybrid Analysis Engine - Sequential execution of both analysis tools
"""

import argparse
import json
import subprocess
import sys
from pathlib import Path
from typing import Dict, List, Tuple, Optional, Any
from dataclasses import dataclass, field
from datetime import datetime
import os


@dataclass
class WorkflowMetadata:
    """Metadata extracted from n8n workflow"""
    filepath: str
    workflow_id: str
    workflow_name: str
    node_count: int
    node_types: Dict[str, int]
    has_connections: bool
    connections_count: int


@dataclass
class ValidationResult:
    """Result of workflow validation"""
    valid: bool
    errors: List[str] = field(default_factory=list)
    warnings: List[str] = field(default_factory=list)


@dataclass
class AnalysisResult:
    """Combined analysis results from both tools"""
    workflow_path: str
    metadata: WorkflowMetadata
    agentic_radar_findings: List[Dict[str, Any]]
    semgrep_findings: List[Dict[str, Any]]
    total_findings: int
    execution_time: float
    timestamp: str


# ============================================================================
# Component A: Input and Validation Module
# ============================================================================

class WorkflowValidator:
    """Validates n8n workflow JSON files and extracts metadata"""

    def __init__(self):
        self.required_fields = ['nodes']

    def load_workflow(self, filepath: str) -> Tuple[Optional[Dict], ValidationResult]:
        """
        Load and parse a workflow JSON file with error handling.

        Args:
            filepath: Path to the workflow JSON file

        Returns:
            Tuple of (workflow_data, validation_result)
        """
        result = ValidationResult(valid=False)

        # Check file exists
        if not os.path.exists(filepath):
            result.errors.append(f"File not found: {filepath}")
            return None, result

        if not os.path.isfile(filepath):
            result.errors.append(f"Path is not a file: {filepath}")
            return None, result

        # Try to read and parse JSON
        try:
            with open(filepath, 'r', encoding='utf-8') as f:
                workflow = json.load(f)
        except json.JSONDecodeError as e:
            result.errors.append(f"Invalid JSON syntax: {e}")
            return None, result
        except IOError as e:
            result.errors.append(f"Cannot read file: {e}")
```

```python
            return None, result

        # Validate structure
        validation = self.validate_structure(workflow)
        if not validation.valid:
            return workflow, validation

        result.valid = True
        return workflow, result

    def validate_structure(self, workflow: Dict) -> ValidationResult:
        """
        Validate that workflow has required structure.

        Args:
            workflow: Parsed workflow dictionary

        Returns:
            ValidationResult with validation status and messages
        """
        result = ValidationResult(valid=True)

        # Check for nodes array
        if 'nodes' not in workflow:
            result.errors.append("Missing required field: 'nodes'")
            result.valid = False
        elif not isinstance(workflow['nodes'], list):
            result.errors.append("Field 'nodes' must be an array")
            result.valid = False
        else:
            # Validate node structure
            for idx, node in enumerate(workflow['nodes']):
                if not isinstance(node, dict):
                    result.warnings.append(f"Node at index {idx} is not an object")
                    continue

                # Check for required node fields
                required_node_fields = ['id', 'name', 'type']
                for field in required_node_fields:
                    if field not in node:
                        result.warnings.append(
                            f"Node at index {idx} missing field '{field}'"
                        )

        # Check for connections (not required, but note if missing)
        if 'connections' not in workflow:
            result.warnings.append("Workflow has no 'connections' field")
        elif not isinstance(workflow['connections'], dict):
            result.warnings.append("Field 'connections' should be an object")

        return result

    def extract_metadata(self, workflow: Dict, filepath: str) -> WorkflowMetadata:
        """
        Extract metadata from workflow.

        Args:
            workflow: Parsed workflow dictionary
            filepath: Path to workflow file

        Returns:
            WorkflowMetadata object
        """
        # Count node types
        node_types = {}
        for node in workflow.get('nodes', []):
            node_type = node.get('type', 'unknown')
            node_types[node_type] = node_types.get(node_type, 0) + 1

        # Count connections
        connections = workflow.get('connections', {})
        connections_count = sum(
            len(conns) for conns in connections.values() if isinstance(conns, dict)
        )

        return WorkflowMetadata(
            filepath=filepath,
            workflow_id=workflow.get('id', 'unknown'),
            workflow_name=workflow.get('name', 'unnamed'),
            node_count=len(workflow.get('nodes', [])),
            node_types=node_types,
            has_connections='connections' in workflow,
            connections_count=connections_count
        )

    def scan_directory(self, directory: str, pattern: str = "*.json") -> List[str]:
        """
        Scan directory for workflow files.

        Args:
            directory: Directory path to scan
            pattern: Glob pattern for file matching (default: *.json)

        Returns:
            List of file paths
        """
        dir_path = Path(directory)
        if not dir_path.exists():
            return []
```

```
        if not dir_path.is_dir():
            return []

        return [str(f) for f in dir_path.glob(pattern)]


# ============================================================================
# Component B: Hybrid Analysis Engine
# ============================================================================

class AgenticRadarExecutor:
    """Executes Agentic Radar security analysis"""

    def __init__(self, output_dir: Optional[str] = None):
        self.output_dir = output_dir or "analysis_output"
        Path(self.output_dir).mkdir(parents=True, exist_ok=True)

    def run(self, workflow_path: str) -> Tuple[List[Dict], bool, str]:
        """
        Execute Agentic Radar scan on workflow.

        Args:
            workflow_path: Path to workflow JSON file

        Returns:
            Tuple of (findings_list, success, error_message)
        """
        workflow_name = Path(workflow_path).stem
        output_path = Path(self.output_dir) / f"{workflow_name}_radar"

        # Construct command
        cmd = [
            "agentic-radar",
            "scan",
            workflow_path,
            "--output",
            str(output_path)
        ]

        try:
            result = subprocess.run(
                cmd,
                capture_output=True,
                text=True,
                timeout=60  # 60 second timeout
            )

            if result.returncode != 0:
                return [], False, f"Agentic Radar failed: {result.stderr}"

            # Parse output - Agentic Radar may produce HTML or JSON
            findings = self._parse_output(output_path, workflow_name)
            return findings, True, ""

        except subprocess.TimeoutExpired:
            return [], False, "Agentic Radar execution timed out"
        except FileNotFoundError:
            return [], False, "Agentic Radar not found. Is it installed?"
        except Exception as e:
            return [], False, f"Agentic Radar execution error: {e}"

    def _parse_output(self, output_path: Path, workflow_name: str) -> List[Dict]:
        """
        Parse Agentic Radar output files.

        Args:
            output_path: Directory containing output files
            workflow_name: Name of the workflow

        Returns:
            List of finding dictionaries
        """
        findings = []

        # Look for JSON output first
        json_file = output_path / f"{workflow_name}.json"
        if json_file.exists():
            try:
                with open(json_file, 'r') as f:
                    data = json.load(f)
                    # Extract findings from JSON structure
                    if isinstance(data, dict) and 'findings' in data:
                        findings = data['findings']
                    elif isinstance(data, list):
                        findings = data
            except Exception as e:
                print(f"Warning: Could not parse Agentic Radar JSON: {e}", file=sys.stderr)

        # If no JSON, look for HTML and note its presence
        html_file = output_path / f"{workflow_name}.html"
        if html_file.exists() and not findings:
            findings.append({
                'tool': 'agentic-radar',
                'type': 'report_generated',
                'message': f'HTML report generated at {html_file}',
                'severity': 'info'
            })
```

```
        return findings


class SemgrepExecutor:
    """Executes Semgrep security analysis with custom n8n rules"""

    def __init__(self, rules_path: Optional[str] = None):
        self.rules_path = rules_path or "rules/n8n-generic-patterns.yaml"

    def run(self, workflow_path: str) -> Tuple[List[Dict], bool, str]:
        """
        Execute Semgrep scan on workflow.

        Args:
            workflow_path: Path to workflow JSON file

        Returns:
            Tuple of (findings_list, success, error_message)
        """
        # Check if rules file exists
        if not os.path.exists(self.rules_path):
            return [], False, f"Semgrep rules not found: {self.rules_path}"

        # Construct command
        cmd = [
            "semgrep",
            "--config", self.rules_path,
            "--json",
            workflow_path
        ]

        try:
            result = subprocess.run(
                cmd,
                capture_output=True,
                text=True,
                timeout=60  # 60 second timeout
            )

            # Semgrep returns 0 or 1 (1 when findings exist), both are success
            if result.returncode not in [0, 1]:
                return [], False, f"Semgrep failed: {result.stderr}"

            # Parse JSON output
            findings = self._parse_output(result.stdout)
            return findings, True, ""

        except subprocess.TimeoutExpired:
            return [], False, "Semgrep execution timed out"
        except FileNotFoundError:
            return [], False, "Semgrep not found. Is it installed?"
        except Exception as e:
            return [], False, f"Semgrep execution error: {e}"

    def _parse_output(self, stdout: str) -> List[Dict]:
        """
        Parse Semgrep JSON output.

        Args:
            stdout: Semgrep stdout containing JSON

        Returns:
            List of finding dictionaries
        """
        try:
            data = json.loads(stdout)
            findings = []

            # Semgrep native JSON format
            if 'results' in data:
                for result in data['results']:
                    findings.append({
                        'tool': 'semgrep',
                        'rule_id': result.get('check_id', 'unknown'),
                        'message': result.get('extra', {}).get('message', result.get('message', '')),
                        'severity': result.get('extra', {}).get('severity', 'WARNING'),
                        'path': result.get('path', ''),
                        'line': result.get('start', {}).get('line', 0),
                        'code': result.get('extra', {}).get('lines', ''),
                        'metadata': result.get('extra', {}).get('metadata', {})
                    })

            return findings

        except json.JSONDecodeError as e:
            print(f"Warning: Could not parse Semgrep JSON: {e}", file=sys.stderr)
            return []


class HybridAnalyzer:
    """Orchestrates hybrid analysis using both Agentic Radar and Semgrep"""

    def __init__(self, agentic_output_dir: Optional[str] = None,
                 semgrep_rules: Optional[str] = None):
        self.validator = WorkflowValidator()
        self.agentic_executor = AgenticRadarExecutor(agentic_output_dir)
        self.semgrep_executor = SemgrepExecutor(semgrep_rules)

    def analyze(self, workflow_path: str) -> Optional[AnalysisResult]:
```

```
        """
        Perform hybrid analysis on a workflow.

        Args:
            workflow_path: Path to workflow JSON file

        Returns:
            AnalysisResult or None if validation fails
        """
        start_time = datetime.now()

        # Load and validate workflow
        print(f"\n[*] Loading workflow: {workflow_path}")
        workflow, validation = self.validator.load_workflow(workflow_path)

        if not validation.valid:
            print(f"[!] Validation failed for {workflow_path}:")
            for error in validation.errors:
                print(f"    ERROR: {error}")
            return None

        if validation.warnings:
            print(f"[!] Validation warnings:")
            for warning in validation.warnings:
                print(f"    WARNING: {warning}")

        # Extract metadata
        metadata = self.validator.extract_metadata(workflow, workflow_path)
        print(f"[*] Workflow metadata:")
        print(f"    Name: {metadata.workflow_name}")
        print(f"    ID: {metadata.workflow_id}")
        print(f"    Nodes: {metadata.node_count}")
        print(f"    Node types: {len(metadata.node_types)}")

        # Execute Agentic Radar (sequential execution)
        print(f"\n[*] Running Agentic Radar analysis...")
        radar_findings, radar_success, radar_error = self.agentic_executor.run(workflow_path)

        if not radar_success:
            print(f"[!] Agentic Radar error: {radar_error}")
            radar_findings = []
        else:
            print(f"[+] Agentic Radar completed: {len(radar_findings)} findings")

        # Execute Semgrep (sequential execution)
        print(f"\n[*] Running Semgrep analysis...")
        semgrep_findings, semgrep_success, semgrep_error = self.semgrep_executor.run(workflow_path)

        if not semgrep_success:
            print(f"[!] Semgrep error: {semgrep_error}")
            semgrep_findings = []
        else:
            print(f"[+] Semgrep completed: {len(semgrep_findings)} findings")

        # Calculate execution time
        end_time = datetime.now()
        execution_time = (end_time - start_time).total_seconds()

        # Create analysis result
        result = AnalysisResult(
            workflow_path=workflow_path,
            metadata=metadata,
            agentic_radar_findings=radar_findings,
            semgrep_findings=semgrep_findings,
            total_findings=len(radar_findings) + len(semgrep_findings),
            execution_time=execution_time,
            timestamp=datetime.now().isoformat()
        )

        return result

    def analyze_batch(self, directory: str) -> List[AnalysisResult]:
        """
        Analyze all workflows in a directory.

        Args:
            directory: Directory containing workflow JSON files

        Returns:
            List of AnalysisResult objects
        """
        workflow_files = self.validator.scan_directory(directory)
        print(f"[*] Found {len(workflow_files)} workflow files in {directory}")

        results = []
        for workflow_file in workflow_files:
            result = self.analyze(workflow_file)
            if result:
                results.append(result)

        return results


# ============================================================================
# Report Generation
# ============================================================================

def generate_json_report(results: List[AnalysisResult], output_path: str):
    """Generate JSON report from analysis results"""
```

```
    report_data = []

    for result in results:
        report_data.append({
            'workflow_path': result.workflow_path,
            'workflow_name': result.metadata.workflow_name,
            'workflow_id': result.metadata.workflow_id,
            'node_count': result.metadata.node_count,
            'node_types': result.metadata.node_types,
            'analysis': {
                'agentic_radar_findings': result.agentic_radar_findings,
                'semgrep_findings': result.semgrep_findings,
                'total_findings': result.total_findings,
                'execution_time': result.execution_time,
                'timestamp': result.timestamp
            }
        })

    with open(output_path, 'w', encoding='utf-8') as f:
        json.dump(report_data, f, indent=2)

    print(f"\n[+] JSON report saved to: {output_path}")


def generate_markdown_report(results: List[AnalysisResult], output_path: str):
    """Generate Markdown report from analysis results"""
    with open(output_path, 'w', encoding='utf-8') as f:
        f.write("# n8n Workflow Security Analysis Report\n\n")
        f.write(f"**Generated:** {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}\n\n")
        f.write(f"**Total workflows analyzed:** {len(results)}\n\n")

        # Summary statistics
        total_findings = sum(r.total_findings for r in results)
        total_radar = sum(len(r.agentic_radar_findings) for r in results)
        total_semgrep = sum(len(r.semgrep_findings) for r in results)

        f.write("## Summary\n\n")
        f.write(f"- Total findings: {total_findings}\n")
        f.write(f"- Agentic Radar findings: {total_radar}\n")
        f.write(f"- Semgrep findings: {total_semgrep}\n\n")

        # Per-workflow results
        for idx, result in enumerate(results, 1):
            f.write(f"## Workflow {idx}: {result.metadata.workflow_name}\n\n")
            f.write(f"**Path:** `{result.workflow_path}`\n\n")
            f.write(f"**Metadata:**\n")
            f.write(f"- Workflow ID: {result.metadata.workflow_id}\n")
            f.write(f"- Node count: {result.metadata.node_count}\n")
            f.write(f"- Execution time: {result.execution_time:.2f}s\n\n")

            # Agentic Radar findings
            f.write(f"### Agentic Radar Findings ({len(result.agentic_radar_findings)})\n\n")
            if result.agentic_radar_findings:
                for finding in result.agentic_radar_findings:
                    f.write(f"- **{finding.get('type','Unknown')}**: {finding.get('message','')}\n")
            else:
                f.write("No findings.\n")
            f.write("\n")

            # Semgrep findings
            f.write(f"### Semgrep Findings ({len(result.semgrep_findings)})\n\n")
            if result.semgrep_findings:
                for finding in result.semgrep_findings:
                    severity = finding.get('severity', 'UNKNOWN')
                    rule = finding.get('rule_id', 'unknown')
                    message = finding.get('message', '')
                    line = finding.get('line', 0)
                    f.write(f"- **[{severity}]** {rule} (line {line}): {message}\n")
            else:
                f.write("No findings.\n")
            f.write("\n")
            f.write("---\n\n")

    print(f"[+] Markdown report saved to: {output_path}")


# ============================================================================
# CLI Interface
# ============================================================================

def main():
    """Main entry point for the workflow analyzer"""
    parser = argparse.ArgumentParser(
        description="n8n Workflow Security Analyzer - Hybrid analysis using Agentic Radar and Semgrep",
        formatter_class=argparse.RawDescriptionHelpFormatter,
        epilog="""
Examples:
  # Analyze single workflow
  %(prog)s workflow_1.json

  # Analyze with custom output directory
  %(prog)s workflow_1.json --output results/

  # Batch analysis
  %(prog)s --batch ./workflows/

  # Generate JSON report
  %(prog)s workflow_1.json --format json --report output.json
        """
```

```
    )

    # Input arguments
    parser.add_argument(
        'workflow',
        nargs='?',
        help='Path to workflow JSON file'
    )
    parser.add_argument(
        '--batch',
        metavar='DIR',
        help='Analyze all workflows in directory'
    )

    # Output arguments
    parser.add_argument(
        '--output',
        metavar='DIR',
        default='analysis_output',
        help='Output directory for analysis results (default: analysis_output)'
    )
    parser.add_argument(
        '--rules',
        metavar='PATH',
        default='rules/n8n-generic-patterns.yaml',
        help='Path to Semgrep rules file (default: rules/n8n-generic-patterns.yaml)'
    )

    # Report arguments
    parser.add_argument(
        '--format',
        choices=['json', 'markdown', 'both'],
        default='markdown',
        help='Report format (default: markdown)'
    )
    parser.add_argument(
        '--report',
        metavar='PATH',
        help='Path for generated report (default: auto-generated in output dir)'
    )

    args = parser.parse_args()

    # Validate input
    if not args.workflow and not args.batch:
        parser.error("Either workflow file or --batch directory must be specified")

    if args.workflow and args.batch:
        parser.error("Cannot specify both workflow file and --batch directory")

    # Initialize analyzer
    analyzer = HybridAnalyzer(
        agentic_output_dir=args.output,
        semgrep_rules=args.rules
    )

    # Perform analysis
    results = []
    if args.batch:
        results = analyzer.analyze_batch(args.batch)
    else:
        result = analyzer.analyze(args.workflow)
        if result:
            results = [result]

    if not results:
        print("\n[!] No successful analyses to report")
        return 1

    # Generate reports
    print("\n" + "="*70)
    print("ANALYSIS COMPLETE")
    print("="*70)

    timestamp = datetime.now().strftime('%Y%m%d_%H%M%S')

    if args.format in ['json', 'both']:
        json_path = args.report if args.report and args.format == 'json' else \
                    f"{args.output}/report_{timestamp}.json"
        generate_json_report(results, json_path)

    if args.format in ['markdown', 'both']:
        md_path = args.report if args.report and args.format == 'markdown' else \
                  f"{args.output}/report_{timestamp}.md"
        generate_markdown_report(results, md_path)

    # Print summary
    print(f"\n{'='*70}")
    print("SUMMARY")
    print("="*70)
    print(f"Workflows analyzed: {len(results)}")
    print(f"Total findings: {sum(r.total_findings for r in results)}")
    print(f"Average execution time: {sum(r.execution_time for r in results) / len(results):.2f}s")

    return 0


if __name__ == '__main__':
    sys.exit(main())
```

B Regras Semgrep para n8n

Arquivo de configuração contendo as 7 regras Semgrep desenvolvidas para detecção de vulnerabilidades em workflows n8n.

Listing 6: rules/n8n-generic-patterns.yaml - Regras de segurança para n8n

```yaml
rules
  - id n8n-pgpassword-injection
    message |
      Detected PGPASSWORD with n8n dynamic expression. Password could contain
      shell metacharacters leading to command injection.
    severity ERROR
    languages
      - generic
    metadata
      category security
      cwe CWE-78
    patterns
      - pattern-regex 'PGPASSWORD.*\{\{.*\}\}'

  - id n8n-ssh-key-exposed
    message |
      Detected hardcoded SSH key. Keys should not be embedded in workflows.
    severity WARNING
    languages
      - generic
    metadata
      category security
      cwe CWE-798
    patterns
      - pattern-regex 'ssh-(rsa|ed25519|dss)\s+[A-Za-z0-9+/=]{50,}'

  - id n8n-command-with-expression
    message |
      SSH or execute command with n8n expression. Risk of command injection.
    severity ERROR
    languages
      - generic
    metadata
      category security
      cwe CWE-78
    patterns
      - pattern-regex '"command":\s*".*\{\{.*\$json.*\}\}'

  - id n8n-hardcoded-password-in-json
    message |
      Hardcoded password found in workflow. Use n8n credentials instead.
    severity ERROR
    languages
      - generic
    metadata
      category security
      cwe CWE-798
    patterns
      - pattern-regex '"password":\s*"[A-Za-z0-9!@#$%^&*]{8,}"'
      - pattern-not-regex '\{\{'

  - id n8n-sql-injection-risk
    message |
      SQL query with dynamic n8n expression. Risk of SQL injection.
    severity ERROR
    languages
      - generic
    metadata
      category security
      cwe CWE-89
    patterns
      - pattern-regex '(SELECT|INSERT|UPDATE|DELETE|FROM|WHERE).*\{\{.*\$json'

  - id n8n-delete-dynamic-url
    message |
      HTTP DELETE with dynamic URL. Risk of unauthorized deletion.
    severity ERROR
    languages
      - generic
    metadata
      category security
      cwe CWE-285
    patterns
      - pattern-regex '"method":\s*"DELETE"'
      - pattern-regex '"url":\s*".*\{\{.*\}\}'

  - id n8n-vault-secret-operation
    message |
      Operation on Vault secrets endpoint. Ensure proper authorization.
    severity WARNING
    languages
      - generic
    metadata
      category security
    patterns
      - pattern-regex 'vault.*/(metadata|data)/.*\{\{.*\}\}'
```