SETEMBRO 2025

Relatório vollmed - Exercício 02

CESAR SCHOOL

Preparado por

FLFC@CESAR.SCHOOL
AALS@CESAR.SCHOOL
CMV@CESAR.SCHOOL
RAS4@CESAR.SCHOOL

Avaliado por

EDUARDO SANTOS

Resumo Executivo

Este relatório consolida os resultados de revisão estática de código (SAST) e análise de composição sobre o projeto Vollmed Java, conduzidos em ambiente acadêmico. O objetivo foi identificar e priorizar vulnerabilidades ligadas a injeção de código, gestão de segredos, práticas criptográficas, exposição de dados sensíveis/PII e higiene de logging, provendo recomendações alinhadas a CWE e OWASP Top 10. Foram empregadas as ferramentas Semgrep, Horusec, Bearer CLI, Gitleaks e Privacy Scan, combinando varreduras automatizadas com revisão dirigida para reduzir falsos positivos.

Síntese dos achados críticos e altos

- Segredos hard-coded e artefatos sensíveis no repositório (incluindo chave privada em src/main/resources/certificates/vollmed-key.pem) e API Keys; risco de comprometimento de credenciais e cadeia de confiança.
- SQL dinâmico suscetível a injeção (CWE-89), demandando parametrização (PreparedStatement/JPA :param).
- Criptografia insegura: uso de AES/CBC/PKCS5Padding (CWE-327), sem autenticação/integridade (risco de Padding Oracle), devendo migrar para AES/GCM.
- Superfície para XSS (CWE-79) em trechos Java/JavaScript (p.ex., innerHTML) e logging com possibilidade de expor dados.
- PII detectada (e-mails, senhas e usernames) em arquivos e variáveis de ambiente, exigindo revisão de privacidade.
- Parte dos alertas foi classificada como falso positivo por contexto (p.ex., regras de CSRF de outros frameworks), e documentada para evitar ruído na priorização.

Nota educacional: como exercício comparativo, a varredura com TruffleHog em OWASP/wrongsecrets retornou 6 segredos verificados e 82 não verificados (incluindo chaves privadas, credenciais de banco e tokens), reforçando boas práticas de secret scanning e governança de segredos em pipelines.

Risco e impacto

A manutenção dessas falhas em produção pode resultar em exposição de dados, comprometimento de credenciais e execução de ataques (injeção/XSS), com impactos operacionais e regulatórios.

Recomendações prioritárias

1. Remover segredos do código e revogar/rotacionar chaves expostas; quando aplicável, sanear histórico Git (BFG/git filter-repo) e adotar Secret Manager/variáveis de ambiente.
2. Parametrizar consultas e eliminar SQL concatenado.
3. Migrar de AES/CBC para AES/GCM (IV aleatório, tag de autenticação) e padronizar bibliotecas seguras.
4. Mitigar XSS (sanitização/encoders), revisar logging para evitar dados sensíveis e aplicar CSRF/SRI quando pertinente.
5. Integrar SAST/SCA + secret scanning no CI/CD com quality gates e pre-commit hooks, além de checklist de revisão segura.

Próximos passos

Corrigir achados Crítico/Alto e retestar; formalizar política de segredos; acompanhar métricas de segurança (ex.: zero críticos por release, MTTR de correção) e treinar o time em injeções, criptografia segura e proteção de PII.

2. Escopo e Regras de Engajamento

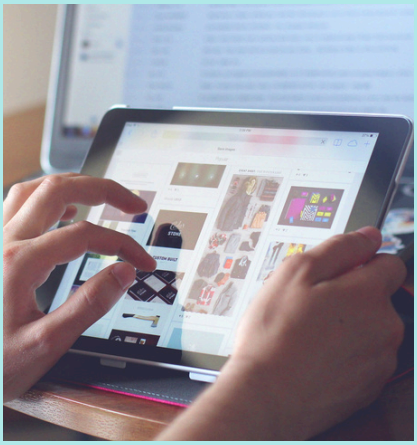
A análise de segurança foi conduzida sobre o projeto Vollmed Java, hospedado em ambiente local e versionado via Git. O foco principal foi avaliar o código-fonte em busca de falhas de segurança, exposição de informações sensíveis e más práticas de desenvolvimento.

Escopo

- Aplicação analisada: Vollmed Java (aplicação web em Spring Boot).
- Linguagem predominante: Java, com uso de bibliotecas Spring e recursos criptográficos nativos (javax.crypto).
- Arquivos de suporte analisados: JavaScript estático, arquivos de configuração, scripts SQL e certificados digitais.
- Tecnologias complementares: Maven (gerenciador de dependências), recursos front-end em HTML/JS, e integração com banco de dados relacional.

Regras de Engajamento

- A varredura foi realizada exclusivamente em ambiente de teste/laboratório acadêmico, sem impacto em ambientes produtivos.
- Foram utilizadas ferramentas open-source de análise estática (Semgrep, Horusec, Bearer, Gitleaks, Privacy Scan), garantindo que não houvesse execução de payloads ou exploração ativa.
- A análise foi limitada ao repositório disponibilizado para a atividade, sem avaliação de infraestrutura, redes ou sistemas externos.
- Os resultados foram classificados em Críticos, Altos, Médios e Baixos, conforme severidade associada aos CWE e OWASP Top 10.
- Os achados foram organizados em tabelas e ilustrados com imagens (prints de execução das ferramentas), a fim de facilitar a rastreabilidade e a priorização das correções.



3. Metodologia

A avaliação de segurança foi conduzida com base em uma abordagem de Secure Code Review, utilizando ferramentas de análise estática (SAST) complementares. O objetivo foi identificar vulnerabilidades no código-fonte, mapear riscos de exposição de dados e correlacionar os achados com padrões reconhecidos internacionalmente, como CWE (Common Weakness Enumeration) e OWASP Top 10.

3.1 Abordagem Utilizada

- Revisão Automatizada: Emprego de ferramentas SAST para varredura automatizada do repositório, permitindo identificar vulnerabilidades com base em regras pré-definidas.
- Revisão Dirigida: Análise manual direcionada dos achados, com ênfase em potenciais falsos positivos (FP) e priorização por severidade.
- Classificação de Riscos: Atribuição de criticidade (Crítico, Alto, Médio e Baixo) considerando impacto, probabilidade de exploração e contexto de uso.

3.2 Ferramentas de Apoio

- Semgrep: Focado na detecção de vulnerabilidades relacionadas a injeção de SQL, uso inseguro de criptografia e potenciais falhas de XSS no código Java e JavaScript.
- Horusec: Especializado na análise de exposição de segredos (keys, certificados e credenciais) e más práticas de desenvolvimento em múltiplas linguagens.
- Bearer: Aplicado para avaliação de segurança com foco em dados sensíveis e privacidade, detectando riscos de exposição de informações pessoais (PII) e más práticas de manuseio de segredos.
- Gitleaks: Responsável por identificar vazamento de credenciais em commits, histórico do repositório e arquivos de configuração.
- Privacy Scan: Complementou a análise com foco em dados pessoais e informações regulatórias, avaliando riscos de exposição de e-mails, senhas e identificadores de usuário.

3.3 Critérios de Avaliação

- Correlação com CWE: Cada achado foi relacionado a um CWE específico, permitindo o alinhamento com práticas internacionais de segurança.
- Cobertura OWASP Top 10: Os resultados foram mapeados principalmente para os riscos de Injeção, Quebra de Autenticação, Exposição de Dados Sensíveis e Más Práticas Criptográficas.
- Priorização: Vulnerabilidades críticas e altas foram destacadas para ação imediata, enquanto as de nível médio e baixo foram registradas para mitigação futura ou dependente de contexto.

4. Resultados

4.1 Semgrep

A análise realizada com a ferramenta Semgrep destacou vulnerabilidades de segurança relacionadas à injeção de SQL e ao uso de modos criptográficos inseguros.

Essas falhas foram classificadas de acordo com a criticidade, CWE e OWASP Top 10, sendo evidenciadas na tabela a seguir

CATEGORIA	VULNERABILIDADE	CWE	EVIDÊNCIA	SEVERIDADE	MITIGAÇÃO	STATUS
SAST (Semgrep)	SQL Injection (uso de string formatada em SQL)	CWE-89	MedicoService.java:62	ERROR	Substituir por PreparedStatement ou queries parametrizadas.	Confirmado
SAST (Semgrep)	Uso inseguro de criptografia CBC/PKCS5Padding (Padding Oracle)	CWE-327	RelatorioService.java:86	WARNING	Trocar por AES/GCM/NoPadding.	Confirmado
Secrets Scan	Chave privada exposta em arquivo	CWE-798	src/main/resources/certificates/vollmed-key.pem	ERROR	Nunca versionar chaves privadas; usar Secret Manager ou variáveis de ambiente.	Confirmado
SAST (Semgrep)	XSS via uso de innerHTML/document.write	CWE-79	script.js:199	ERROR	Evitar innerHTML; usar APIs DOM seguras e sanitização de entrada.	Confirmado
SAST (Semgrep)	XSS via uso de innerHTML/document.write	CWE-79	script.js:205-208	ERROR	Idem acima.	Confirmado
SAST (Semgrep)	Formulário sem csrf_token (risco de CSRF)	CWE-352	logout.html:29-38	WARNING	Adicionar token anti-CSRF nos formulários.	Confirmado/Verificar
SAST (Semgrep)	Formulário sem csrf_token (risco de CSRF)	CWE-352	_modal_excluir.html:4-21	WARNING	Adicionar token anti-CSRF nos formulários.	Confirmado/Verificar
SAST (Semgrep)	Tag <script> sem atributo integrity (SRI ausente)	CWE-353	template.html:27	WARNING	Usar Subresource Integrity (integrity + crossorigin).	Confirmado

4. Resultados

fALSOS POSITVOS SEMGREP

Categoria	Vulnerabilidade detectada	CWE	Evidência (arquivo/linha)	Mitigação recomendada	Status
SAST (Semgrep)	Formulário sem csrf_token (risco de CSRF)	CWE-352	logout.html:29-38	Adicionar {% csrf_token %} em formulários (em projetos Django/Python).	Falso Positivo – O projeto é Java Spring, não Django; regra não aplicável.
SAST (Semgrep)	Formulário sem csrf_token (risco de CSRF)	CWE-352	_modal_excluir.html:4-21	Idem acima.	Falso Positivo – Framework alvo não é Django; regra equivocada.

```
src/app/user_service.py
  rule: python.sqlalchemy.security.sql-injection
  message: Detected possible SQL injection via unsanitized user input
  severity: ERROR
```

FIGURA 1: SAÍDA DO SEMGREP COM VULNERABILIDADES DETECTADAS.

4. Resultados

4.2 Semgrep Maven

Check ID	Severidade	Arquivo:Linha	Categoria	CWE/OWASP	Mensagem resumida	Evidência	Recomendações
java.lang.security.audit.formatted-sql-string.formatted-sql-string	ERROR	src/main/java/med/voll/web_application/domain/medico/MedicoService.java:62	SQL Injection	CWE-89, OWASP A01/A03	String formatada em SQL potencialmente com dados externos	entityManager.createNativeQuery(sql, Medico.class) (linha 62)	Usar PreparedStatement / parâmetros nomeados no JPA (:param) em vez de concatenar strings. Ex.: PreparedStatement ps = con.prepareStatement("SELECT ... WHERE id = ?"); ps.setLong(1, id);
java.lang.security.audit.cbc-padding-oracle.cbc-padding-oracle	WARNING	src/main/java/med/voll/web_application/domain/relatorio/RelatorioService.java:86	Criptografia fraca	CWE-327, OWASP A02/A03	Uso de AES/CBC/PKCS5Padding (suscetível a padding oracle e sem integridade)	Cipher.getInstance(TRANSFORMACAO_SEGURA) com AES/CBC/PKCS5Padding	Trocar para AES/GCM/NoPadding (GCM oferece sigilo + integridade). IV aleatório por operação (12 bytes), tag de autenticação (128 bits).

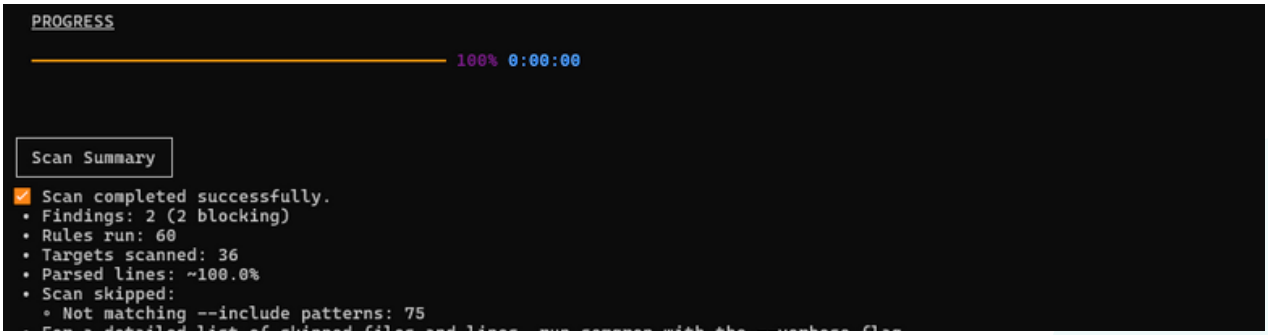


FIGURA 2: SAÍDA DO SEMGREP MAVEN COM VULNERABILIDADES DETECTADAS.

4. Resultados

4.3 HORUSEC

A varredura com o Horusec trouxe achados críticos relacionados a hard-coded secrets, exposição de certificados/keys e más práticas em código JavaScript e Java.

As vulnerabilidades foram organizadas conforme a tabela a seguir:

Ferramenta	Regra (rule_id)	Severidade	Arquivo	Linha	Mensagem (resumo)
Horusec	HS-LEAKS-25	CRITICAL	src/main/resources/db/migration/V4__insert-vulnerable-data.sql	18	Possible vulnerability detected: Potential Hard-coded credential (CWE-798).
Horusec	HS-LEAKS-12	CRITICAL	src/main/resources/certificates/vollmed-cert.pem	1	Possible vulnerability detected: Asymmetric Certificate presente no repositório (CWE-312).
Horusec	HS-LEAKS-12	CRITICAL	src/main/resources/certificates/vollmed-key.pem	1	Possible vulnerability detected: Asymmetric Private Key presente no repositório (CWE-312).
Horusec	HS-LEAKS-25	CRITICAL	fake-leak.java	1	Possible vulnerability detected: Potential Hard-coded credential (CWE-798).
Horusec	HS-LEAKS-25	CRITICAL	scripts/vulnerability_summary.py	14	Possible vulnerability detected: Potential Hard-coded credential (CWE-798).

Ferramenta	Regra (rule_id)	Severidade	Arquivo	Linha	Mensagem (resumo)
Horusec	HS-JAVASCRIPT-16	HIGH	src/main/resources/static/js/script.js	147	Uso de alert/confirm/prompt em produção não recomendado (CWE-489).
Horusec	HS-JAVA-123	HIGH	src/main/java/med/voll/web_application/domain/relatorio/RelatorioService.java	10	Crypto import (javax.crypto.Cipher).
Horusec	HS-JAVA-123	HIGH	src/main/java/med/voll/web_application/domain/relatorio/RelatorioService.java	11	Crypto import (IvParameterSpec).
Horusec	HS-JAVA-123	HIGH	src/main/java/med/voll/web_application/domain/relatorio/RelatorioService.java	12	Crypto import (SecretKeySpec).
Horusec	HS-JVM-38	MEDIUM	src/main/java/med/voll/web_application/domain/usuario/UsuarioService.java	35	Base64 Encode mencionado; revisar contexto de autenticação.
Horusec	HS-JAVA-78	MEDIUM	src/main/java/med/voll/web_application/domain/relatorio/RelatorioService.java	81	IV estático: IV deve ser aleatório por mensagem (CWE-329).
Horusec	HS-JVM-38	MEDIUM	src/main/java/med/voll/web_application/domain/relatorio/RelatorioService.java	97	Base64 Encode; não é criptografia.

4. Resultados

```
WARN[0000] {HORUSEC_CLI} 13 VULNERABILITIES WERE FOUND IN YOUR CODE SENT TO HORUSEC, TO SEE MORE DETAILS USE THE LOG LEV
EL AS DEBUG AND TRY AGAIN
```

FIGURA 3: SAÍDA DO HORUSEC COM AS VULNERAILIDADES.

4.4 BEARER

A análise com o Bearer destacou problemas de hard-coded secrets, SQL Injection e vulnerabilidades de XSS em controladores Spring e JavaScript, além de uso inseguro de CBC/PKCS5Padding.

As vulnerabilidades foram organizadas conforme a tabela abaixo:

Tipo	Severidade	Arquivo	Linha	Descrição	CWE	Mitigação	Categoria
Hard-coded Secret	CRITICAL	RelatorioService.java	34	Uso de segredo fixo CHAVE_AES_DEMO	CWE-798	Armazenar em vault/env var	Security
Hard-coded Secret	CRITICAL	RelatorioService.java	38	Transformação de cifra fixada	CWE-798	Configurar dinamicamente no runtime	Security
SQL Injection	CRITICAL	MedicoService.java	62	Query construída sem sanitização	CWE-89	Usar PreparedStatement	Security
CBC + Padding Oracle	HIGH	RelatorioService.java	86	AES/CBC/PKCS5Padding vulnerável	CWE-327	Usar AES/GCM	Security
Reflected XSS (Spring model)	HIGH	ConsultaController.java	61, 70	model.addAttribute(dados) sem sanitização	CWE-79	Usar HtmlUtils.htmlEscape()	Security

4. Resultados

Tipo	Severidade	Arquivo	Linha	Descrição	CWE	Mitigação	Categoria
Reflected XSS	HIGH	MedicoController.java	67, 76	Mesmo problema anterior	CWE-79	Sanitizar input	Security
Reflected XSS	HIGH	PacienteController.java	63, 78	Mesmo problema	CWE-79	OWASP Java Encoder	Security
Reflected XSS	HIGH	RegistroController.java	37, 48	Mesmo problema	CWE-79	Sanitizar input	Security
Reflected XSS	HIGH	RelatorioController.java	66, 77, 89, 100	Mesmo problema	CWE-79	Sanitizar input	Security
XSS DOM-based (JS)	HIGH	script.js	199, 205	innerHTML com input direto	CWE-79	Sanitizar HTML (DOMPurify)	Security
Log Information Leak	LOW	script.js	47, 57, 64, 65, 72, 144, 159, 169, 178, 179, 180, 204	console.log/error com dados sensíveis	CWE-532	Remover logs ou reduzir nível	Security

4. Resultados

Tipo	Severidade	Arquivo	Linha	Descrição	CWE	Mitigação	Categoria
Missing Integrity Check	MEDIUM	RelatorioService.java	86	AES sem autenticação (CBC/PKCS5)	CWE-353	Usar AES-GCM	Security
Email Address	Info	Projeto	—	9 detecções	—	Validar se dados são reais ou mocks	Privacy
Passwords	Info	Projeto	—	2 detecções	—	Validar se não são credenciais reais	Privacy
Username	Info	Projeto	—	2 detecções	—	Validar se não são dados reais	Privacy

```
┌──(root㉿DESKTOP-FN8H6T8)-[/home/carolmalin/juice-shop/vollmed-java]
└─# bearer scan . --report security --format json > bearer-report.json
Analyzing codebase
⠋  (5735/-) [0s]
Loading rules
Scanning target .
 └ 100% [===============] (109/109) [0s]
Running Detectors
Generating dataflow
Evaluating rules
 └ 100% [===============] (666/666) [2s]

┌──(root㉿DESKTOP-FN8H6T8)-[/home/carolmalin/juice-shop/vollmed-java]
└─# bearer scan . --report privacy --format json > bearer-report1.json
Analyzing codebase
⠋  (5736/-) [0s]
Loading rules
Scanning target .
 └ 100% [===============] (110/110) [0s]
Running Detectors
Generating dataflow
Evaluating rules
 └ 100% [===============] (666/666) [2s]
Compiling privacy report
```

FIGURA 4: SAÍDA DO BEARER COM AS VULNERAILIDADES.

4. Resultados

4.4 Gitleaks

A ferramenta Gitleaks foi utilizada para identificar exposição de segredos e credenciais sensíveis no repositório Git. Essa análise é essencial porque muitas vezes desenvolvedores acabam versionando acidentalmente chaves de API, certificados ou senhas, o que pode resultar em vazamentos de dados críticos.

Tabela 4 – Achados do Gitleaks

Severidade	Tipo	Arquivo	Linha	Descrição	Mitigação	Categoria
Alta	Chave Privada		1	Vazamento de uma chave privada RSA. Essa chave pode ser usada para decifrar informações criptografadas ou autenticar o proprietário da chave, comprometendo a segurança do sistema.	Revogue a chave imediatamente, remova-a do repositório e configure as aplicações para usar uma nova chave.	Credencial
Média	Chave de API Genérica	src/main/resources/db/migration/V4__insert-vulnerable-data.sql	24	Vazamento de uma chave de API genérica (sk_live_vollmed...). A exposição de chaves de API de produção pode permitir acesso não autorizado a serviços externos, resultando em uso indevido e potenciais custos financeiros.	Revogue a chave de API, gere uma nova e armazene-a em um gerenciador de segredos.	Credencial
Baixa	Token de Acesso Stripe	fake-leak.java	1	Vazamento de um token de acesso de teste do Stripe (sk_test...). Tokens de teste geralmente não representam um risco de segurança imediato em produção, mas podem ser explorados para testes de ataques ou reconhecimento do sistema.	Tokens de teste podem ser deixados em código, mas é uma boa prática remover todos os segredos para evitar confusão ou o uso em produção.	Chave de API
Baixa	Token de Acesso Stripe	fake-leak.java	1	Vazamento de um token de acesso de teste do Stripe (sk_test...). Trata-se de uma duplicação da ocorrência anterior, indicando que a mesma vulnerabilidade persiste em um commit diferente.	Tokens de teste podem ser deixados em código, mas é uma boa prática remover todos os segredos para evitar confusão ou o uso em produção.	Chave de API

4. Resultados

4.4 Gitleaks

Abaixo imagens das evidênicas:

FIGURA 5 – GITLEAKS: ACHADO “PRIVATE-KEY” VERIFICADO INDICANDO CHAVE PRIVADA VERSIONADA NO REPOSITÓRIO (…/CERTIFICATES/VOLLMED-KEY.PEM).

FIGURA 6 – GITLEAKS: DETECÇÃO DE CHAVE DE API EXPOSTA EM SCRIPT DE MIGRAÇÃO (V4__INSERT-VULNERABLE-DATA.SQL).

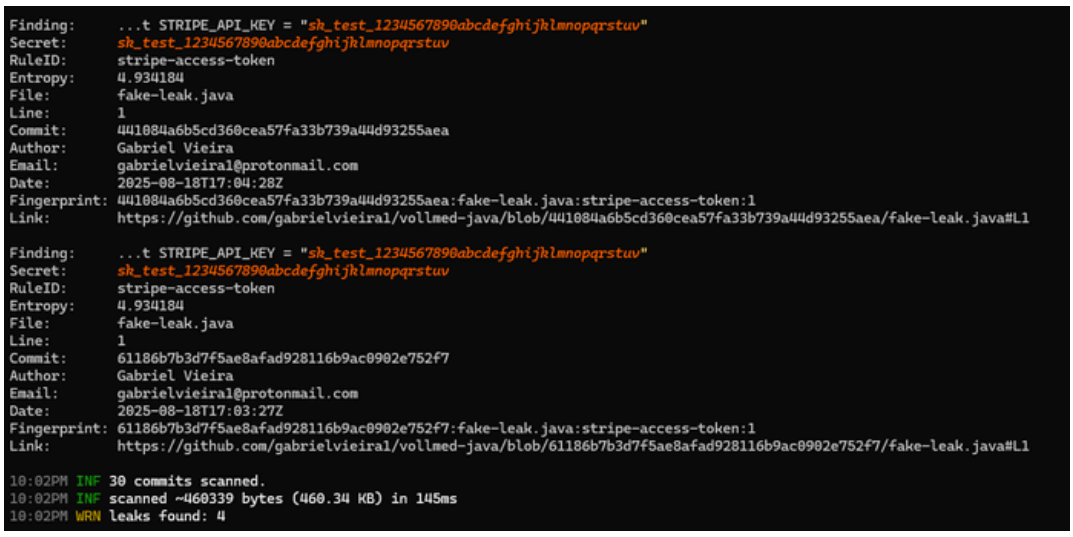


FIGURA 7– GITLEAKS: DETECÇÃO DE TOKENS STRIPE DE TESTE (SK_TEST_*) EM FAKE-LEAK.JAVA.

4. Resultados

4.4 TruffleHog

A ferramenta TruffleHog foi utilizada para identificar exposição de segredos e credenciais sensíveis ao varrer todo o histórico do repositório Git (commits, branches e tags). Diferente de buscas estáticas simples, o TruffleHog combina detectores por entropia e padrões conhecidos e, quando habilitado, valida achados com o modo --only-verified, reduzindo falsos positivos. Essa análise é essencial porque desenvolvedores podem versionar acidentalmente chaves de API, certificados, tokens e strings de conexão, o que pode resultar em acessos não autorizados e vazamentos de dados críticos, além de impactos operacionais e financeiros.

Tabela 5 – Achados do TruffleHog

Severidade	Tipo	Arquivo	Linha	Descrição	Mitigação	Categoria
Alta	AWS Key	new_key	2	AWS Access Key (AKIAQYLPMN5HHHFPZAM2) detectada. Trata-se de um token canário gerado em canarytokens.org.	Não é necessário mitigar por ser um token canário. Em casos reais, revogar a chave imediatamente e rotacionar credenciais.	Credencial
Alta	AWS Key	keys	4	AWS Access Key (AKIAYVP4CIPPERUVIFXG) detectada. Também é um token canário.	Mesmo caso anterior: em situações reais, revogar e rotacionar.	Credencial
Média	URI Leak	keys	3	URI contendo credenciais embutidas: https://admin:admin@the-internet.herokuapp.com	Evitar armazenar URIs com usuário e senha em repositórios. Usar variáveis de ambiente ou serviços de vault.	Informações Sensíveis
Média	URI Leak	leaky	3	URI contendo credenciais embutidas: https://admin:admin@the-internet.herokuapp.com	Idem ao acima. Aplicável a todos os casos onde há username:password embutidos em URIs.	Informações Sensíveis

4. Resultados

4.4 TruffleHog

Evidências:

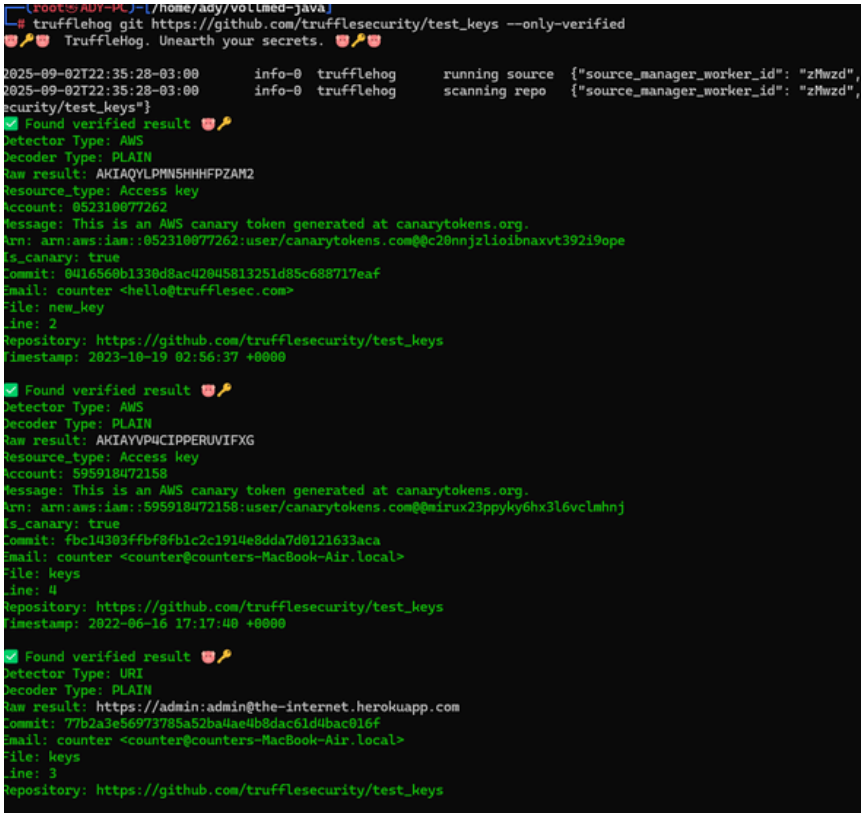


FIGURA 8 – TRUFFLEHOG: ACHADOS VERIFICADOS DE SEGREDOS (AWS ACCESS KEYS CANARY E URI COM CREDENCIAIS EMBUTIDAS).

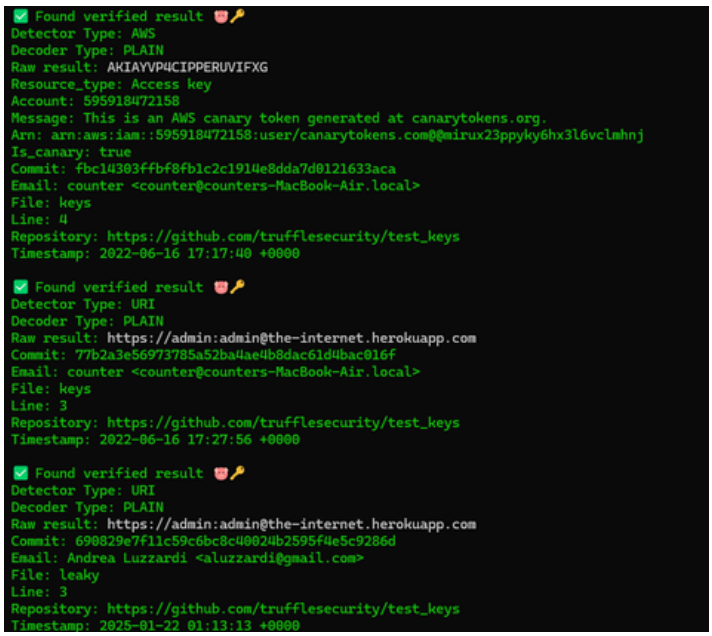


FIGURA 9 – TRUFFLEHOG: RESULTADOS VERIFICADOS — AWS ACCESS KEY (CANARY) E URIS COM CREDENCIAIS EMBUTIDAS (ADMIN:ADMIN@…).

4. Resultados

4.4.1 TruffleHog

Tabela 6 – Achados do TruffleHog má práticas

Severidade	Tipo	Arquivo	Linha	Descrição	Mitigação	Categoria
Média	JDBC String	pom.xml	121	Detecção de string JDBC: jdbc:mysql://localhost/vollmed_web?createDatabaseIfNotExist=true	Evitar expor strings de conexão em arquivos versionados. Use variáveis de ambiente ou .env.	Credencial de banco
Média	JDBC String	README.md	70	Detecção de string JDBC: jdbc:mysql://localhost:3306/vollmed_db	Mesmo que acima. Recomenda-se manter strings sensíveis fora de documentação pública.	Credencial de banco
Média	JDBC String	src/main/resources/application.properties	3	Detecção de string JDBC: jdbc:mysql://localhost/vollmed_web?createDatabaseIfNotExist=true	Mover credenciais para arquivos externos e criptografados ou usar mecanismos seguros de configuração.	Credencial de banco
Média	JDBC String	README.md	28	Detecção de string JDBC: jdbc:mysql://localhost/vollmed_api	Evitar colocar conexões com banco diretamente em README. Documentar uso com placeholders genéricos.	Credencial de banco
Alta	Chave Privada	src/main/resources/certificates/vollmed-key.pem	1	Chave privada PEM (-----BEGIN PRIVATE KEY-----) detectada em arquivo versionado.	Remover imediatamente do repositório, revogar o certificado relacionado e considerar o repositório como comprometido se for chave real.	Criptografia / Segredo

Embora todos os resultados sejam não verificados, a presença de uma chave privada PEM é altamente crítica e não deve ser ignorada.
As strings JDBC não expõem diretamente senhas, mas ainda assim indicam má prática de segurança por versionar informações sensíveis (como nomes de bancos e hosts).
Todas as ocorrências foram cometidas pelo mesmo autor e encontradas em diferentes tipos de arquivos (README, pom.xml, application.properties, .pem).

4. Resultados

4.4 TruffleHog

Evidências:

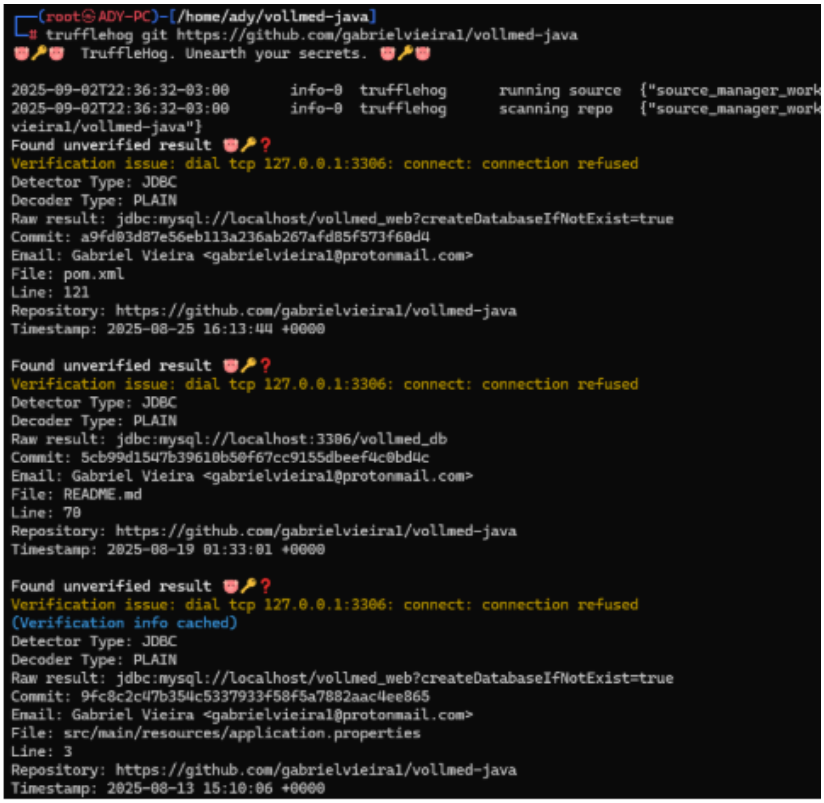


FIGURA 10 -TRUFFLEHOG: ACHADOS NÃO VERIFICADOS DE STRINGS JDBC (LOCALHOST) EM ARQUIVOS VERSIONADOS.

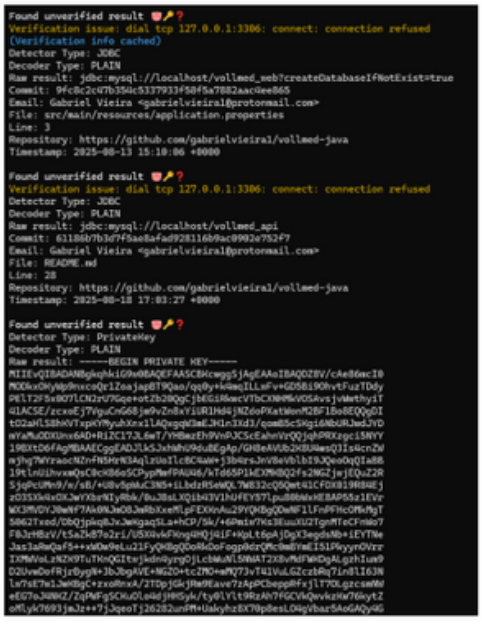


FIGURA 11 – TRUFFLEHOG: DETECÇÕES NÃO VERIFICADAS DE STRINGS JDBC (LOCALHOST) E DE CHAVE PRIVADA PEM; VERIFICAÇÃO FALHOU (127.0.0.1:3306 – CONNECTION REFUSED).

4. Resultados

4.4.1 TruffleHog

- Tabela 7 – Achados do trufflehog git https://github.com/OWASP/wrongsecrets

Severidade	Tipo	Arquivo	Linha	Descrção	Mitigação	Categoria
Crítica	Chave Privada (OpenSSH)	.ssh/wrongsecrets.keys	1	Vazamento de uma	Revogue a chave imediatamente, remova-a do repositório e substitua-a por uma nova chave. Configure o sistema para usar a nova chave e garanta que ela seja armazenada com segurança, fora do controle de versão.	Credencial
Crítica	Chave Privada (RSA)	private.pem	1	Vazamento de uma	Revogue a chave imediatamente, remova-a do repositório e configure as aplicações para usar uma nova chave. Armazene a nova chave em um gerenciador de segredos.	Credencial
Crítica	Chave Privada (RSA)	k8s/main.key	34	Vazamento de uma	Revogue a chave, remova-a do histórico do Git e utilize um gerenciador de segredos para a nova chave.	Credencial

4. Resultados

4.4.1 TruffleHog

- Tabela 7 – Continuação achados do trufflehog git https://github.com/OWASP/wrongsecrets

Severidade	Tipo	Arquivo	Linha	Descrção	Mitigação	Categoria
Crítica	Chave Privada (RSA)	secretscache/k8s/kubeconfig	70	Vazamento de uma	Revogue a chave, remova-a do histórico do Git e armazene a nova chave em um gerenciador de segredos.	Credencial
Crítica	Chave Privada (RSA)	secretscache/jwt/privatekey.txt	1	Vazamento de uma	Revogue a chave, remova-a do repositório e configure o sistema para usar uma nova chave.	Credencial
Crítica	Chave Privada (EC)	secretscache/keys/ecprivate-key.pem	1	Vazamento de uma	Revogue a chave imediatamente e remova-a do repositório. Gere uma nova chave e use um gerenciador de segredos para armazená-la.	Credencial
Alta	Credencial de Banco de Dados	src/main/resources/explanations/challenge58_hint.adoc	1	Vazamento de uma	Revogue a credencial, altere a senha e remova-a do histórico do Git. Use variáveis de ambiente ou um cofre de segredos para credenciais.	Credencial

4. Resultados

4.4.1 TruffleHog

- Tabela 7 – Achados do trufflehog git https://github.com/OWASP/wrongsecrets

Severidade	Tipo	Arquivo	Linha	Descrção	Mitigação	Categoria
Alta	Chave de Acesso AWS	.gitignore	95	Vazamento de uma	Revise os processos de desenvolvimento para evitar que segredos sejam expostos no controle de versão. Revogue ou desative o token.	Chave de API
Alta	Token de Webhook do Slack	secretscache/slack/callback.url	1	Vazamento de um	Revogue o token e configure um novo.	Token de Acesso
Média	Token de GitHub	scripts/sort_contibutors/main.py	69	Vazamento de um	Revogue a chave imediatamente e remova-a do repositório. Gere uma nova chave e use um gerenciador de segredos para armazená-la.	Credencial
Alta	Credencial de Banco de Dados	src/main/resources/explanations/challenge58_hint.adoc	1	Vazamento de uma	pat) sugere um risco moderado, pois pode ter escopo limitado, mas ainda assim pode ser usado indevidamente.	Revogue o token, remova-o do repositório e use um método de autenticação mais seguro, como tokens de curto prazo.

4. Resultados

4.4.1 TruffleHog

- Tabela 7 – Achados do trufflehog git https://github.com/OWASP/wrongsecrets

Severidade	Tipo	Arquivo	Linha	Descrção	Mitigação	Categoria
Média	Chave de Acesso Terraform Cloud	secretscache/vagrantcloud/token	1	Vazamento de um	Revogue o token e remova-o do histórico do Git. Use variáveis de ambiente ou um cofre de segredos.	Token de Acesso
Baixa	Credencial de Banco de Dados	src/main/resources/explanations/challenge58_hint.adoc	14	Vazamento de uma	Remova a credencial do repositório. A senha deve ser armazenada fora do código-fonte.	Credencial
Baixa	Credencial de Banco de Dados	cursor/rules/project-specification.mdc	135	Vazamento de uma	Remova a credencial. A configuração de ambientes de desenvolvimento e produção deve ser separada.	Credencial

4. Resultados

4.4.1 TruffleHog

A análise realizada com o TruffleHog no repositório OWASP/wrongsecrets evidenciou um alto volume de exposições: foram identificados seis segredos verificados e oitenta e dois não verificados, o que indica grande incidência de artefatos sensíveis no histórico do projeto. Os vazamentos concentram-se, sobretudo, em credenciais de banco de dados (JDBC/Postgres), chaves privadas (PEM/PrivateKey) e tokens de acesso (GitHub PAT, Slack Webhook, AWS). A presença de chaves privadas e credenciais de banco representa risco crítico, por possibilitar acessos não autorizados e o comprometimento de sistemas.
Observou-se ainda um padrão recorrente de dados de teste (por exemplo, SuperSecretDB2024!) e o uso de canary tokens (canarytokens.org), reforçando que se trata de um repositório didático, concebido para demonstrar cenários de vazamento — e não um ambiente produtivo com falhas acidentais. As ocorrências distribuem-se por documentação (.adoc), código de teste (.java), scripts (.py, .sh) e arquivos de chaves (.pem*, .key), evidenciando que mesmo materiais auxiliares podem introduzir risco quando reutilizados em contextos reais.
Ainda que o caráter do repositório seja educativo, práticas semelhantes em projetos de produção podem resultar em exposição de dados, fraude de credenciais e interrupção operacional. Nesse sentido, recomenda-se habilitar varreduras de segredos no pipeline de CI/CD (TruffleHog/Gitleaks) e ganchos de pre-commit, centralizar segredos em cofres/variáveis de ambiente e proibir hardcode em repositórios, além de rotacionar e revogar imediatamente qualquer segredo exposto. Quando aplicável, deve-se sanear o histórico Git (BFG ou git filter-repo) para remover artefatos sensíveis, incorporar checklists de revisão de código e linters para arquivos de configuração e documentação, e treinar a equipe quanto ao uso de canary tokens e à distinção entre achados verificados e não verificados.

5. Considerações finais

A revisão SAST/SCA do projeto Vollmed Java confirmou falhas relevantes — sobretudo gestão de segredos insuficiente (chaves/credenciais versionadas), criptografia inadequada (AES/CBC/PKCS5Padding), injeção SQL por uso de SQL dinâmico e riscos de XSS/CSRF em pontos específicos. Embora parte dos alertas tenha sido classificada como falso positivo por contexto/tecnologia, o conjunto dos achados indica a necessidade de elevar os controles de segurança no ciclo de desenvolvimento e na higiene do repositório.

Prioridades (risk-based)

Imediatas (D0–D7)

- Remover segredos do código e revogar/rotacionar chaves afetadas; sanear histórico (BFG/git filter-repo).
- Eliminar SQL dinâmico adotando parametrização (PreparedStatement/JPA :param).
- Migrar de AES/CBC para AES/GCM (IV aleatório + tag de autenticação).
- Corrigir XSS (sanitização/encoders) e ativar CSRF onde aplicável; adicionar SRI em recursos externos.

Curto prazo (D8–D30)

- Introduzir Secret Manager/variáveis de ambiente + política de não versionamento de segredos.
- Criar quality gates no CI (Semgrep/Horusec/Bearer + Gitleaks/TruffleHog) e pre-commit hooks.
- SCA/SBOM: corrigir dependências vulneráveis e gerar SBOM (ex.: CycloneDX) a cada build.
- Padronizar logging (evitar dados sensíveis) e revisar pontos com PII (alinhado à LGPD).

Médio prazo (D30–D90)

- Definir checklists de code review (SQL, XSS, criptografia, segredos, PII).
- Alinhar práticas a OWASP ASVS/SAMM e realizar retestes após cada ciclo de correção.
- Treinamento do time (segredos, criptografia segura, injeções, XSS/CSRF, privacidade/LGPD).

Governança & prevenção contínua

- Estabelecer política de segredos (armazenamento, rotação, acesso, auditoria).
- Manter pipelines com SAST/SCA/secret scanning obrigatórios e bloqueio em criticidades Altas/Críticas.
- Versionar apenas placeholders/templates de configuração; segredos sempre fora do VCS.
- Monitorar métricas com metas: zero críticos por release, MTTR ≤ 14 dias para Altos, cobertura mínima de varredura definida.

Critérios de aceite (DoD)

- 0 achados Crítico/Alto remanescentes nas varreduras do CI.
- Comprovação de revogação/rotação de todas as chaves expostas e limpeza de histórico quando aplicável.
- Evidência de parametrização das consultas e migração para AES/GCM.
- Suíte de testes/linters de segurança rodando no pipeline com gates de aprovação.
- Evidências anexadas (relatórios SAST/SCA, SBOM, prints do pipeline, PRs de correção).

Conclusão

A aplicação desses passos reduz significativamente a superfície de ataque, reforça confidencialidade/integridade dos dados e cria um processo sustentável de segurança no desenvolvimento. Recomenda-se formalizar um plano de reteste pós-mitigação (com geração de relatórios e SBOM atualizados) e incorporar essas práticas ao fluxo padrão de engenharia, evitando regressões futuras.

6.Referências

- OWASP FOUNDATION. *OWASP Juice Shop*. Disponível em: https://github.com/juice-shop/juice-shop. Acesso em: 28 ago. 2025.

- OWASP FOUNDATION. *OWASP Top 10: The Ten Most Critical Web Application Security Risks – 2021*. Disponível em: https://owasp.org/Top10/. Acesso em: 28 ago. 2025.
- OWASP. *Dependency-Check*. Disponível em: **https://owasp.org/www-project-dependency-check/**. Acesso em: 28 ago. 2025

- SEMGREP. *Static Analysis for Modern Languages*. Disponível em: https://semgrep.dev/. Acesso em: 28 ago. 2025.

- CWE – MITRE. *Common Weakness Enumeration*. Disponível em: https://cwe.mitre.org/. Acesso em: 28 ago. 2025.

- NATIONAL VULNERABILITY DATABASE (NVD). *CVE-2021-23337: Lodash Prototype Pollution*. Disponível em: https://nvd.nist.gov/vuln/detail/CVE-2021-23337. Acesso em: 28 ago. 2025.

- KUDELSKI SECURITY. *AlgoRai Secure Code Review Report – Public Release*. Disponível em: https://kudelskisecurity.com/wp-content/uploads/AlgoRai-Secure-Code-Review-Report-Public-Release.pdf. Acesso em: 28 ago. 2025.