Universidade de Brasília – UnB
Faculdade UnB Gama – FGA
Engenharia de Software

# DevSecOps: Um Estudo sobre a Aplicação de Segurança ao Pipeline DevOps de um Software

Autor: Dafne Moretti Moreira e Lucas da Cunha Andrade
Orientadora: Profa. Dra. Milene Serrano
Coorientador: Prof. Dr. Maurício Serrano

Brasília, DF
2024

Dafne Moretti Moreira e Lucas da Cunha Andrade

# DevSecOps: Um Estudo sobre a Aplicação de Segurança ao Pipeline DevOps de um Software

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Universidade de Brasília – UnB

Faculdade UnB Gama – FGA

Orientador: Profa. Dra. Milene Serrano

Coorientador: Prof. Dr. Maurício Serrano

Brasília, DF

2024

Dafne Moretti Moreira e Lucas da Cunha Andrade

# DevSecOps: Um Estudo sobre a Aplicação de Segurança ao Pipeline DevOps de um Software

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Trabalho aprovado. Brasília, DF, 08 de Julho de 2024:

---

**Profa. Dra. Milene Serrano**
Orientadora

---

**Prof. Dr. Maurício Serrano**
Coorientador

---

**Prof. Dr. Renato Coral Sampaio**
Examinador

---

**Thaiane Ferreira Braga**
**Consultora Desenvolvedora**
**Thoughtworks**
Examinadora

Brasília, DF
2024

# Agradecimentos

Eu, Dafne, agradeço primeiramente a Deus, que me concedeu força e discernimento durante esta etapa da minha jornada. Essa conquista, devo também à minha família, pois sem eles não seria possível. À minha mãe, Andréia, que sempre me encorajou a acreditar nos meus sonhos. Aos meus irmãos, Adne e Pedro, por tornarem a minha vida mais leve durante este período. Ao meu pai, Davi, por me ensinar o valor da disciplina. Em especial ao meu padrinho, Andrey, por ser a minha inspiração, me dar suporte para seguir este caminho, e mesmo de longe contribuir de forma imprescindível para que eu chegasse até aqui. À minha tia Cristiane, pelos cafés depois de longos períodos de estudo que foram momentos de grande alegria. Aos meus primos, Gabriel e Vitor, por todo o apoio. À minha madrinha, Leondina, pelas ligações descontraídas em momentos difíceis na graduação.

Agradeço a todos os amigos presentes durante a graduação, tornando os dias mais leves e dividindo as frustrações e sucessos comigo. Principalmente à minha dupla, Lucas, por compartilhar não só o conhecimento, mas também a amizade e o companheirismo.

Eu, Lucas, agradeço a todos que contribuíram para a realização deste trabalho. Agradeço profundamente à minha mãe, pelo seu amor incondicional e por sempre me apoiar em todos os desafios que tive, tanto durante a elaboração deste trabalho, quanto durante toda a graduação. Sua força, dedição e apoio foram fundamentais para este momento.

Agradeço a minha dupla Dafne, que me auxiliou durante a elaboração deste trabalho. Muito obrigado pela parceria e colaboração. Todos os momentos de esforço conjunto. Mesmo nos momentos mais tensos, agradeço pelo apoio, e sem esse apoio, não seria possível alcançar este resultado

Juntos, agradecemos também aos nossos orientadores, Dra. Milene Serrano e Dr. Maurício Serrano, por toda a orientação, paciência e apoio durante este percurso. Sua dedicação e ajuda foi fundamental para a conclusão deste trabalho.

A esses, nossa eterna gratidão!

# Resumo

A segurança de software é um conjunto de ações, processos e técnicas utilizado para garantir um sistema protegido de acessos não autorizados a ativos e recursos. Torna-se, assim, fundamental que a segurança seja implementada em todo o ciclo de vida de desenvolvimento de software. Contudo, muitos produtos de software utilizados não são seguros, seja por negligências ao longo do desenvolvimento dos mesmos; seja por não continuidade de investimentos no que compete ao critério de segurança após esses produtos serem implantados. Por isso, a implementação da segurança no *pipeline* DevOps (Desenvolvimento e Operações), ou seja, no processo que promove a colaboração e a integração entre as equipes de desenvolvimento e operações do software, garante monitoramento contínuo de vulnerabilidades e resposta rápida a possíveis ataques de segurança. A OWASP Top 10 apresenta uma série de diretrizes a práticas a serem seguidas que evitam as vulnerabilidades mais encontradas, inclusive para o *pipeline Continuous Integration/Continuous Delivery* (CI/CD), que significam, respectivamente, Integração e Entrega Contínuas. Com base na premissa de que a segurança na atividade de gerência de configuração do software garante um ambiente de produção mais seguro, este trabalho apresenta um *pipeline* DevSecOps. Cabe ressaltar que a atividade de gerenciamento de configuração é realizada de forma transversal no ciclo de vida de um software, com maior intensidade nas etapas de desenvolvimento e manutenção evolutiva de software. Por fim, entende-se que DevSecOps, abreviação de desenvolvimento, segurança e operações, automatiza a integração do critério de segurança nas etapas do ciclo de vida de desenvolvimento de software, desde o *design* inicial até a entrega e a evolução do software.

**Palavras-chave**: DevSecOps, Segurança de Software, OWASP, DevOps, Cibersegurança, Integração e Entrega Contínuas

# Abstract

Software security encompasses a set of actions, processes, and techniques designed to safeguard a system against unauthorized access to assets and resources. Therefore, it is essential to implement security measures throughout the software development lifecycle. However, numerous software pieces lack security, either due to negligence during development or a failure to sustain investments in security criteria post-implementation. This is where integrating security into the DevOps (Development and Operations) pipeline becomes crucial. The DevOps pipeline fosters collaboration and integration between development and operations teams, ensuring continuous monitoring of vulnerabilities and swift responses to potential security threats. The OWASP Top 10 provides guidelines and practices to prevent commonly encountered vulnerabilities, including those in the Continuous Integration/Continuous Delivery (CI/CD) pipeline, representing Continuous Integration and Continuous Delivery, respectively. Building on the premise that security in software configuration management ensures a more secure production environment, this paper presents a DevSecOps pipeline. It's worth noting that configuration management is a vital aspect of the software lifecycle, with greater emphasis during development and maintenance stages. Finally, DevSecOps, an abbreviation for development, security, and operations, automates the incorporation of security criteria into various stages of the software development lifecycle. This integration spans from the initial design phase to the delivery and ongoing evolution of the software.

**Key-words**: DevSecOps, Software Security, OWASP, DevOps, Cybersecurity, Integration and Continuous Delivery

# Lista de ilustrações


Figura 1 – Ciclo de vida de software com etapas relacionadas à segurança . . . . . 33
Figura 2 – Mudanças ocorridas de 2017 para 2021 . . . . . 36
Figura 3 – Exemplo de ataque por quebra de controle de acesso . . . . . 37
Figura 4 – Exemplo de ataque de falha de criptografia . . . . . 38
Figura 5 – Exemplo de ataque de injeção . . . . . 39
Figura 6 – Exemplo de ataque realizado a partir de um *design* inseguro . . . . . 40
Figura 7 – Exemplo de ataque realizado a partir de uma configuração errada . . . 41
Figura 8 – Exemplo de ataque realizado em aplicação utilizando sistemas vulneráveis 42
Figura 9 – Exemplo de ataque realizado com falhas no processo de identificação e autenticação . . . . . 43
Figura 10 – Ataque realizado por falhas de integridade do software . . . . . 44
Figura 11 – Exemplo de ataque realizado por falhas de registro e monitoramento do segurança . . . . . 45
Figura 12 – Exemplo de ataque de falsificação de solicitação do lado do servidor . . 46
Figura 13 – Integração Contínua e Entrega Contínua no *pipeline* DevOps . . . . . 49
Figura 14 – Fases da Segurança Contínua . . . . . 51
Figura 15 – *Guideline* DevSecOps . . . . . 52
Figura 16 – Exemplo de definição de fluxo de trabalho no *GitHub Actions* . . . . . 60
Figura 17 – *Quality Gate* e análise do código fonte no SonarQube . . . . . 63
Figura 18 – Exemplo de OWASP *Dependency-Check* . . . . . 65
Figura 19 – Exemplo de código de ataque SQLMAP . . . . . 67
Figura 20 – Exemplo de resultado do ataque com SQLMAP . . . . . 67
Figura 21 – Método geral orientado a provas de conceito . . . . . 74
Figura 22 – Método de desenvolvimento . . . . . 75
Figura 23 – Fluxo de atividades e subprocessos TCC1 . . . . . 79
Figura 24 – Fluxo de atividades e subprocessos TCC2 . . . . . 81
Figura 25 – Fluxo DevSecOps . . . . . 87
Figura 26 – *Secrets* da organização . . . . . 89
Figura 27 – *Secrets Frontend* . . . . . 89
Figura 28 – *Secrets Backend* . . . . . 90
Figura 29 – Configurações de segurança do Github . . . . . 90
Figura 30 – Configurações avançadas de segurança no Github . . . . . 91
Figura 31 – Aba de Segurança no Github . . . . . 91
Figura 32 – Exemplo de *workflows* bem sucedidos e com falha no GitHub . . . . . 96
Figura 33 – Pasta com data de relatório . . . . . 100
Figura 34 – Exemplo de relatório *.json* produzido pelas ferramentas DAST . . . . . 101

Figura 35 – Exemplo de *Release* . . . . . 102
Figura 36 – Página de Cadastro da Aplicação de Gerenciamento de Senhas . . . . . 104
Figura 37 – Página de *Login* da Aplicação de Gerenciamento de Senhas . . . . . 104
Figura 38 – Página de Listagem das Credenciais da Aplicação de Gerenciamento de Senhas . . . . . 105
Figura 39 – Componentes vulneráveis identificados pelo *npm* no *Frontend* da POC1 108
Figura 40 – Componentes vulneráveis identificados pelo *npm* no *Backend* da POC1 108
Figura 41 – Alertas do OWASP *Dependency Check* no *Frontend* da POC1 . . . . . 109
Figura 42 – Alertas do OWASP *Dependency Check* no *Backend* da POC1 . . . . . 110
Figura 43 – Relatório Snyk no *Frontend* . . . . . 110
Figura 44 – Comentário do SonarCloud no PR do *Frontend* da POC1 . . . . . 111
Figura 45 – Detalhes do SonarCloud no *Frontend* da POC1 . . . . . 111
Figura 46 – Comentário do SonarCloud no PR do *Backend* da POC1 . . . . . 112
Figura 47 – Comentário do SonarCloud no PR do *Frontend* da POC1 . . . . . 112
Figura 48 – Relatório de Inspeção de Identificação de Ausência de *Logs* na POC1 . 113
Figura 49 – Relatório de Inspeção de Identificação de Senha fraca na POC1 . . . . 113
Figura 50 – Relatório de Inspeção de Identificação de Senha em texto claro na POC1 114
Figura 51 – Tipo *any* . . . . . 114
Figura 52 – Resultados do ZAP *Baseline Scan* na POC1 . . . . . 115
Figura 53 – Resultados do ZAP *Full Scan* na POC1 . . . . . 115
Figura 54 – Execução da *action* do Nikto na POC1 . . . . . 116
Figura 55 – Modelagem de Ameaças - Diagrama . . . . . 117
Figura 56 – Modelagem de Ameaças - *Web* . . . . . 117
Figura 57 – Modelagem de Ameaças - Servidor . . . . . 118
Figura 58 – Modelagem de Ameaças - Banco de Dados . . . . . 118
Figura 59 – Página de *Login* da Aplicação Bancária . . . . . 124
Figura 60 – Página de Cadastro da Aplicação Bancária . . . . . 125
Figura 61 – Página principal da Aplicação Bancária . . . . . 125
Figura 62 – Componentes vulneráveis identificados pelo *npm* no *Frontend* da POC2 128
Figura 63 – Componentes vulneráveis identificados pelo *npm* no *Backend* da POC2 129
Figura 64 – Alertas do OWASP *Dependency Check* no *Frontend* da POC2 . . . . . 129
Figura 65 – Alertas do OWASP *Dependency Check* no *Backend* da POC2 . . . . . 130
Figura 66 – Comentário do SonarCloud no PR do *Frontend* DA POC2 . . . . . 130
Figura 67 – Detalhes do SonarCloud no *Frontend* da POC2 . . . . . 131
Figura 68 – Comentário do SonarCloud no PR do *Backend* da POC2 . . . . . 131
Figura 69 – Comentário do SonarCloud no PR do *Backend* da POC2 . . . . . 131
Figura 70 – Ausência de MFA . . . . . 132
Figura 71 – Relatório de Inspeção de Identificação de Ausência de *Logs* na POC2 . 132

Figura 72 – Relatório de Inspeção de Identificação Identificação de Criptografia fraca na POC2 . . . 133
Figura 73 – Relatório de Inspeção de Identificação de Senha fraca na POC2 . . . 133
Figura 74 – Relatório de Inspeção de Identificação Aumento de limite do cartão sem verificação na POC2 . . . 134
Figura 75 – Resultados *Baseline Scan* ZAP na POC2 . . . 134
Figura 76 – Resultados *Full Scan* ZAP na POC2 . . . 135
Figura 77 – Execução da *action* do Nikto na POC2 . . . 135
Figura 78 – Modelagem de Ameaças - Diagrama . . . 136
Figura 79 – Modelagem de Ameaças - Web . . . 136
Figura 80 – Modelagem de Ameaças - Servidor . . . 136
Figura 81 – Modelagem de Ameaças - Banco de Dados . . . 137
Figura 82 – Gráfico de Vulnerabilidades por Categoria . . . 144
Figura 83 – Vulnerabilidades extras por ferramentas . . . 145
Figura 84 – Tempo de *Deploy* para Homologação e testes DAST no *Frontend* da POC1 . . . 149
Figura 85 – Tempo de *Deploy* para Homologação e testes DAST no *Frontend* da POC2 . . . 149
Figura 86 – Tempo de *Deploy* em Produção do *Frontend* da POC1 . . . 150
Figura 87 – Tempo de execução do *pipeline* por etapas . . . 150
Figura 88 – Resultados do *pip-audit* no SIGE . . . 152
Figura 89 – Resultados do OWASP Zap no SIGE . . . 154
Figura 90 – Resultados do Nikto no SIGE . . . 155
Figura 91 – A01 Controle de Acesso OWASP ZAP . . . 171
Figura 92 – A01 Controle de Acesso OWASP ZAP Resposta . . . 171
Figura 93 – Email de autorização de testes no SIGE . . . 173

# Lista de tabelas

Tabela 1 – Ferramentas de apoio ao trabalho . . . . . 68
Tabela 2 – Quantidade de vulnerabilidades encontradas pelo *pipeline* . . . . . 122
Tabela 3 – Vulnerabilidades extras . . . . . 122
Tabela 4 – Vulnerabilidades extras na POC2 . . . . . 141
Tabela 5 – Vulnerabilidades verificadas pelo *Pipeline* . . . . . 143
Tabela 6 – Vulnerabilidades extras encontradas pelo *Pipeline* em cada POC . . . . 145
Tabela 7 – Tempo de execução do *Workflow depcheck.yaml* . . . . . 148

# Lista de quadros

Quadro 1 – Classificação da pesquisa . . . . . 69
Quadro 2 – Cronograma de atividades - primeira etapa do TCC . . . . . 82
Quadro 3 – Cronograma de atividades - segunda estapa do TCC . . . . . 83
Quadro 4 – Vulnerabilidades encontradas pelo *pipeline* . . . . . 120
Quadro 5 – Vulnerabilidades encontradas pelo *pipeline* . . . . . 121
Quadro 6 – Vulnerabilidades encontradas pelo *pipeline* . . . . . 121
Quadro 7 – Vulnerabilidades implantadas encontradas pela *pipeline* . . . . . 139
Quadro 8 – Vulnerabilidades encontradas pela *pipeline* . . . . . 140
Quadro 9 – Vulnerabilidades encontradas pela *pipeline* . . . . . 141

# Lista de Códigos


Código 1 – *Docker-compose.yml* do *Frontend* da POC1 . . . . . 88
Código 2 – Configurações do arquivo `build_sast.yaml` . . . . . 92
Código 3 – Auditoria . . . . . 93
Código 4 – Instalação de dependências . . . . . 93
Código 5 – *Action* Snyk . . . . . 94
Código 6 – *Action* OWASP *Dependency Check* . . . . . 94
Código 7 – *Step* do SonarCloud . . . . . 95
Código 8 – Código *Upload* SARIF e Artefatos . . . . . 96
Código 9 – Código do *Baseline Scan* ZAP . . . . . 98
Código 10 – Código do *Full Scan* ZAP . . . . . 99
Código 11 – Código Nikto . . . . . 99
Código 12 – Código para salvar relatórios no *Github* . . . . . 100
Código 13 – SQL *Injection* . . . . . 126
Código 14 – Código do *pip-audit* . . . . . 152
Código 15 – Código do *safety* . . . . . 153
Código 16 – Código *Upload* OWASP ZAP . . . . . 153
Código 17 – Código do Nikto . . . . . 154

# Lista de abreviaturas e siglas

| | |
|---|---|
| API | *Application Programming Interface* |
| CI/CD | *Continuous Integration/Continuous Delivery* |
| CIA | Confiabilidade, Integridade e Disponibilidade |
| CORS | *Cross-Origin Resource Sharing* |
| CVE | *Common Vulnerabilities and Exposures* |
| CWE | *Common Weakness Enumeration* |
| CVSS | *Common Vulnerability Scoring System* |
| DAST | *Dynamic Application Security Testing* |
| DevOps | *Development and Operations* |
| DevSecOps | *Development, Security and Operations* |
| DHS | *Department of Homeland Security* |
| HTTPS | *Hypertext Transfer Protocol Secure* |
| IaC | *Infrastructure as Code* |
| JWT | *JSON Web Token* |
| MVP | *Minimum Viable Product* |
| OWASP | *Open Web Application Security Project* |
| SAST | *Static Application Security Testing* |
| SCA | *Software Composition Analysis* |
| SQL | *Structured Query Language* |
| TCC | Trabalho de Conclusão de Curso |
| TI | Tecnologia da Informação |
| UnB | Universidade de Brasília |

# Sumário


**1 INTRODUÇÃO . . . . . 25**
**1.1 Contextualização . . . . . 25**
**1.2 Justificativa . . . . . 27**
**1.3 Questões de Pesquisa e Desenvolvimento . . . . . 28**
**1.4 Objetivos . . . . . 28**
1.4.1 Objetivo Geral . . . . . 28
1.4.2 Objetivos Específicos . . . . . 28
**1.5 Organização da Monografia . . . . . 29**

**2 REFERENCIAL TEÓRICO . . . . . 31**
**2.1 Considerações Iniciais do Capítulo . . . . . 31**
**2.2 Segurança de Software . . . . . 31**
2.2.1 Testes de Segurança no Ciclo de Vida de Desenvolvimento de Software . . . 34
**2.3 OWASP . . . . . 35**
2.3.1 OWASP Top Ten . . . . . 35
**2.4 DevOps . . . . . 46**
2.4.1 Integração Contínua (CI) . . . . . 49
2.4.2 Entrega Contínua (CD) . . . . . 49
**2.5 DevSecOps . . . . . 50**
**2.6 Considerações Finais do Capítulo . . . . . 55**

**3 SUPORTE TECNOLÓGICO . . . . . 57**
**3.1 Considerações Iniciais do Capítulo . . . . . 57**
**3.2 Ferramentas de Apoio ao Desenvolvimento . . . . . 57**
3.2.1 GitHub . . . . . 57
3.2.2 Visual Studio Code 1.84 . . . . . 57
3.2.3 ReactJS 18.2.0 . . . . . 58
3.2.4 NodeJS 20.9.0 . . . . . 58
3.2.5 ExpressJS 4.18.2 . . . . . 58
3.2.6 MaterialUI . . . . . 59
3.2.7 Bootstrap . . . . . 59
**3.3 Ferramentas de Apoio ao *Pipeline* DevSecOps . . . . . 59**
3.3.1 Criação de CI/CD . . . . . 59
3.3.1.1 GitHub Actions . . . . . 59
3.3.2 Vercel . . . . . 61
3.3.3 Heroku . . . . . 61

3.3.3.1 Docker . . . 61
3.3.4 Ferramentas SAST . . . 62
3.3.4.1 SonarQube . . . 62
3.3.4.2 CodeQL . . . 64
3.3.5 Ferramentas SCA . . . 64
3.3.5.1 Owasp-Dependency-Check 9.0.0 . . . 65
3.3.5.2 Snyk *Open Source* . . . 66
3.3.5.3 Dependabot . . . 66
3.3.5.4 *GitGuardian* . . . 66
3.3.6 Ferramentas DAST . . . 66
3.3.6.1 ZAP 2.14.0 . . . 66
3.3.6.2 SQLMap 1.7 . . . 67
3.3.6.3 Nikto 2.1.5 . . . 68
**3.4 Considerações Finais do Capítulo . . . 68**

**4 METODOLOGIA . . . 69**
**4.1 Considerações Iniciais do Capítulo . . . 69**
**4.2 Classificação de Pesquisa . . . 69**
4.2.1 Abordagem da Pesquisa . . . 69
4.2.2 Natureza da Pesquisa . . . 70
4.2.3 Objetivos da Pesquisa . . . 70
4.2.4 Procedimentos da Pesquisa . . . 71
**4.3 Método Investigativo . . . 71**
**4.4 Método Geral Orientado a Provas de Conceito . . . 72**
**4.5 Método de Desenvolvimento . . . 74**
**4.6 Método de Análise de Resultados . . . 76**
4.6.1 Métricas para Análise do *Pipeline* DevSecOps . . . 76
4.6.2 Análise de Resultados com Pesquisa-Ação . . . 78
**4.7 Fluxos de Atividades . . . 78**
4.7.1 Atividades e Subprocessos - Primeira Etapa do TCC . . . 79
4.7.2 Atividades e Subprocessos - Segunda Etapa do TCC . . . 81
**4.8 Cronograma de Atividades . . . 82**
**4.9 Considerações Finais do Capítulo . . . 83**

**5 PIPELINE DEVSECOPS . . . 85**
**5.1 Considerações Iniciais do Capítulo . . . 85**
**5.2 Contextualização . . . 85**
**5.3 Pipeline DevSecOps . . . 86**
**5.4 Provas de Conceito . . . 102**
5.4.1 POC 1 - Gerenciador de Senhas . . . 102

5.4.1.1 Definição da PoC . . . 102
5.4.1.2 Solução . . . 107
5.4.1.3 Análise de resultados . . . 120
5.4.2 POC 2 - Aplicação Bancária . . . 123
5.4.2.1 Definição da PoC . . . 123
5.4.2.1.1 Vulnerabilidades OWASP Top10 em Estudo . . . 125
5.4.2.2 Solução . . . 128
5.4.2.3 Análise de Resultados . . . 138
**5.5 Considerações Finais do Capítulo . . . 142**

**6 ANÁLISE DE RESULTADOS . . . 143**
**6.1 Considerações Iniciais do Capítulo . . . 143**
**6.2 Resultados . . . 143**
6.2.1 Vulnerabilidades Identificadas . . . 143
6.2.2 Perfil de Risco Crítico . . . 145
6.2.3 Top Tipos de Vulnerabilidades . . . 146
6.2.4 *Runtime* . . . 147
6.2.5 Análise Qualitativa . . . 151
6.2.6 Contribuições do *Pipeline* em Comunidades *Open Source* . . . 151
**6.3 Considerações Finais do Capítulo . . . 155**

**7 CONCLUSÃO . . . 157**
**7.1 Considerações Iniciais do Capítulo . . . 157**
**7.2 Questões de Pesquisa e Desenvolvimento . . . 157**
**7.3 Objetivos Geral e Específicos . . . 158**
**7.4 Trabalhos Futuros . . . 159**
**7.5 Considerações Finais do Capítulo . . . 160**

**REFERÊNCIAS . . . 161**

**APÊNDICES 169**

**APÊNDICE A – DEBATE SOBRE IDENTIFICAÇÃO DE CONTROLE DE ACESSO NA FERRAMENTA ZAP . . . 171**

**APÊNDICE B – AUTORIZAÇÃO PARA TESTES DE SEGURANÇA NO SIGE . . . 173**

# 1 Introdução

Neste capítulo, são apresentados a CONTEXTUALIZAÇÃO, que introduz informações acerca da necessidade de integração da área de segurança nos campos de desenvolvimento e operação, abordando, portanto, aspectos de DevSecOps (*Development, Security and Operations*); a JUSTIFICATIVA para a produção deste trabalho; as QUESTÕES DE PESQUISA E DESENVOLVIMENTO que norteiam o trabalho e foram respondidas pelo mesmo; e os OBJETIVOS, OBJETIVO GERAL e OBJETIVOS ESPECÍFICOS, cumpridos com a realização do mesmo. Por fim, é apontada a ORGANIZAÇÃO DA MONOGRAFIA em capítulos.

## 1.1 Contextualização

Com o propósito de aprimorar antigas metodologias de desenvolvimento de software que se caracterizavam por modelos de ciclo de vida sequenciais (i.e. apenas deve-se avançar para a próxima etapa do ciclo quando a etapa anterior estiver sido concluída) e representadas pelo modelo cascata, surgiu, em meados da década de 90, os "Métodos Leves" (*Lightweight Methods*), onde existe um foco maior em entendimento, disciplina e habilidade (COCKBURN, 2003). Posteriomente, esses métodos foram chamados de "Métodos ágeis", com o lançamento do manifesto ágil em 2001 (BECK et al., 2001). Este movimento teve como objetivo a modernização e a melhoria do processo de desenvolvimento de projetos de software. Em contraposição ao modelo cascata, o ágil considera que mudanças são esperadas, e que portanto, é preciso minimizar as dificuldades causadas por elas. Para isso, o ciclo de desenvolvimento ocorre de forma iterativa e incremental. Assim, o desenvolvimento é modularizado em pequenas partes, reduzindo o esforço necessário para se planejar cada parte. Eventualmente, essas pequenas partes irão compor um Mínimo Produto Viável (em inglês, MVP (*Minimum Viable Product*)), que consiste em uma versão do produto que deve ser usável e com o minímo de funcionalidades para o usuário, sendo que esse poderá fornecer *feedbacks*, aprimorando e evoluindo o produto (RIES, 2009).

Os métodos ágeis tiveram uma adoção consideravável, com empresas reportando que notaram redução de custos, maior produtividade, melhor qualidade e satisfação empresarial (TECHNOLOGIES, 2010). Adicionalmente, foram reportadas dificuldades para adequação dos princípios ágeis (TECHNOLOGIES, 2010). De toda forma, a adoção de métodos ágeis representou uma mudança na condução de projetos, em grande parte, no fluxo de desenvolvimento, onde novas versões do software eram frequentemente libera-

das. Esse aumento no fluxo incorreu em uma relação não harmoniosa entre as equipes de software, responsáveis pelo desenvolvimento, e as equipes de operações, responsáveis pela migração e pela atualização das mudanças feitas pela equipe de software.

No intuito de mitigar esse problema, originou-se o que se conhece como a cultura DevOps. O nome DevOps tem como origem a colaboração eficaz entre as equipes de desenvolvimento (*DEVelopment*) e de operações (*OPerationS*), utilizando desenvolvimento automatizado, implantação e monitoramento de infraestrutura (EBERT et al., 2016). O DevOps busca assim melhorar a interação entre essas duas equipes, fundamentais para o desenvolvimento do projeto. Para isso, recursos de software foram utilizados visando automatizar esse processo, facilitando o rastreamento das fases de desenvolvimento. Com a dinâmica de mercado atual, e a utilização do DevOps, as equipes implantam mudanças centenas ou até milhares de vezes por dia, dependendo do porte da aplicação (KIM et al., 2021).

Segundo Myrbakken; Colomo-Palacios (2017), uma terceira equipe, fundamental na construção de produtos de software robustos, não foi capaz de acompanhar a agilidade e a velocidade proporcionadas pelo DevOps: a equipe de segurança.

A disponibilização de produtos de software seguros é uma preocupação da Engenharia de Software (BOURQUE; FAIRLEY, 2014). O *design* para segurança concentra-se em monitorar divulgação, criação, alteração e exclusão, ou até negação de acesso à informação e a outros recursos, além de vigiar ataques ou violações (BOURQUE; FAIRLEY, 2014). Assim, a área de segurança de rede e de Internet apresenta medidas para garantir confidencialidade, integridade e disponibilidade de recursos dos sistemas (STALLINGS, 2017).

O DevSecOps, que interliga as áreas de Desenvolvimento e Operação com a área de Segurança, aborda diversas práticas de segurança com a finalidade de melhorar a confiabilidade de um software sem deixar de oferecer recursos, correções e atualizações frequentes de software de forma ágil (BOWEN; HASH; WILSON, 2006). Quando a segurança é integrada como parte do DevOps, os controles de segurança ficam diretamente no produto e não incorporados a ele após o mesmo implantado de fato (VEHENT, 2018).

A Open Worldwide Application Security Project (OWASP Foundation, 2001b) é uma organização sem fins lucrativos focada em conceber e manter aplicações confiáveis. Apresenta-se como um padrão sobre riscos de segurança mais críticos para aplicações Web, sendo comumente utilizado como guia para as práticas de segurança implementadas em *pipelines* DevSecOps.

## 1.2 Justificativa

Enquanto o DevOps permitiu uma maior colaboração entre as equipes de desenvolvimento e de operações, a equipe de segurança ficou isolada. A equipe de segurança de software tradicionalmente atua na fase final do projeto, quando a fase de desenvolvimento já terminou. Para isso, são utilizados testes que verificam o comportamento do software sob certas condições. Esse trabalho de elaboração de código seguro e testagem tem como característica ser bastante manual, demorado, custoso e negligenciado pelas empresas. Em um movimento como o DevOps, caracterizado pelas suas agilidade e automação, esse atraso é uma fonte de problemas (MYRBAKKEN; COLOMO-PALACIOS, 2017).

O alto custo e a demora, somados à ideia equivocada de que segurança não é tão importante, resultaram no abandono parcial, e às vezes absoluto, da etapa de testagem. Um recente estudo realizado pela Allied Universal registrou que, somente no ano de 2022, empresas perderam o total de 1 trilhão de doláres com problemas relacionados à segurança (UNIVERSAL, 2023).

Neste contexto, dada a dificuldade das equipes tradicionais de segurança, foi proposta uma evolução no movimento DevOps: o DevSecOps. Com ele, há a introdução de mecanismos automatizados de segurança. Nesse cenário, cabe mencionar sobre as ferramentas capazes de detectar vulnerabilidades na aplicação, inclusive fornecendo ao desenvolvedor um detalhamento da origem da ameaça, facilitando sua correção, e permitindo que o desenvolvedor esteja mais ciente do problema. O DevSecOps permite, assim, uma fase de testes rápida, escalável e eficaz na identificação de erros, facilitando a criação de políticas visando evitar a ocorrência de novos erros no futuro. Segundo Myrbakken; Colomo-Palacios (2017) esses benefícios proporcionados pelo DevSecOps têm como resultado final a redução do tempo gasto, dos riscos e dos custos.

Diante do exposto, com a elaboração desse trabalho, foram realizados levantamentos na comunidade de Engenharia de Software e afins, identificando práticas e ferramentas, bem como implementações e testagens orientadas a esses levantamentos em um software piloto. Proveu-se insumos, utilizando conceitos e técnicas de DevSecOps, que buscam mitigar problemas de cibersegurança, em tempo de desenvolvimento e testagem. Com isso, gerou-se insumos para reduzir, por exemplo, riscos frequentemente encontrados na própria implantação do software (ex. injeção de código e bibliotecas vulneráveis (OWASP Foundation, 2001b)).

## 1.3 Questões de Pesquisa e Desenvolvimento

Dada a importância de cibersegurança, e da necessidade de se pensar nela desde o desenvolvimento de um software, esse trabalho demandou transpor desafios de viés de pesquisa, mas também de cunho aplicado, ou seja, de desenvolvimento. Portanto, seguem as Questões de Pesquisa e a Questão de Desenvolvimento, respondidas por este trabalho:

- Questão_de_Pesquisa_01: Quais são as principais práticas de cibersegurança recomendadas pela comunidade especializada?
- Questão_de_Pesquisa_02: Quais são as principais ferramentas *open source* que viabilizam testar as práticas de cibersegurança antes mesmo da implantação do software?
- Questão_de_Desenvolvimento: Como essas práticas de cibersegurança podem ser incluídas na fase de desenvolvimento de um software, e testadas usando ferramentas *open source*?

## 1.4 Objetivos

No intuito de responder de forma adequada as questões de pesquisa e de desenvolvimento, foram especificados alguns objetivos, de escopos geral e específicos. Seguem seções dedicadas ao Objetivo Geral e aos Objetivos Específicos.

### 1.4.1 Objetivo Geral

O objetivo geral deste trabalho foi a demonstração, utilizando Provas de Conceito, de como práticas de cibersegurança podem ser incluídas na fase de desenvolvimento e testadas via ferramentas *open source*, antes mesmo da implantação do software.

### 1.4.2 Objetivos Específicos

Com o objetivo geral em mente, foi possível definir alguns objetivos específicos no intuito de cumpri-lo adequadamente. Os objetivos específicos definidos para este trabalho foram:

- Objetivo Específico 01: estudo sobre métodos ágeis, mais especificamente no que compreende o surgimento do DevOps, tendo em vista que, ao se orientar por ci-

bersegurança, o *modus operandi* desses métodos, considerando Desenvolvimento e Operações, não pode ser prejudicado;

- Objetivo Específico 02: levantamento sobre práticas de cibersegurança com o intuito de compreender não apenas quais são essas práticas, mas também seus benefícios e suas preocupações quando incorporadas à fase de desenvolvimento;
- Objetivo Específico 03: levantamento de ferramentas, de preferência *open source*, para utilizar no *pipeline* desenvolvimento/teste a fim de identificar ferramentas capazes de detectar diversos tipos de vulnerabilidades antes mesmo da implantação do software em si;
- Objetivo Específico 04: implementação de cenários de uso em um software, orientando-se pelo estudo e pelos levantamentos realizados anteriormente, com o intuito de conferir insumos de cunho mais prático ao *pipeline* DevSecOps. Entende-se por cenários de uso, nesse caso e fazendo um adaptação com base em Silva e Alves (2018), compreendendo a especificação de situações pontuais, vistas como problemas de cibersegurança, e que demandam ser tratadas no software, e
- Objetivo Específico 05: análise dos resultados obtidos visando reportar cada aspecto observado ao longo do processo. Utilizando uma abordagem híbrida: ora quantitativa, com revelação de quantas vulnerabilidade foram identificadas orientando-se pelo *pipeline* DevSecOps; ora qualitativa, com revelação de benefícios, preocupações, dentre outros aspectos não quantificáveis. Demais detalhes serão apresentados no Capítulo 4 - Metodologia.

## 1.5 Organização da Monografia

A monografia está dividida nos capítulos:

- Capítulo 2 - Referencial Teórico: apresenta a fundamentação deste trabalho, abordando conceitos de DevOps, Segurança de Software e DevSecOps, visando compreender o papel da Segurança no contexto de desenvolvimento ágil e teste;
- Capítulo 3 - Suporte Tecnológico: descreve as principais tecnologias utilizadas na elaboração deste trabalho;
- Capítulo 4 - Metodologia: classifica a pesquisa; detalha os métodos utilizados para a condução dos processos investigativo, de desenvolvimento e de análise de resultados, além de conferir temporalidade para realização das atividades, apresentando cronogramas;

- Capítulo 5 - *Pipeline* DevSecOps: discute acerca do *pipeline* desenvolvido, discorrendo em detalhes o funcionamento das ferramentas *open source* nas provas de conceito.
- Capítulo 6 - Análise de Resultados: confere os resultados obtidos, orientando-se pelo método de Análise de Resultados definido, bem como apontando insumos quantitativos e qualitativos, e
- Capítulo 7 - Conclusão: confere as conclusões do trabalho, retomando as questões de pesquisa e desenvolvimento, bem como os objetivos alcançados ao longo do trabalho, com apresentação de comprobatórios.

# 2 Referencial Teórico

## 2.1 Considerações Iniciais do Capítulo

Neste capítulo, dado o objetivo da elaboração e da criação de um ambiente DevSecOps, apresenta-se o referencial teórico deste trabalho. Inicia-se com a apresentação de conceitos de SEGURANÇA DE SOFTWARE. Em seguida, são apresentadas as vulnerabilidades mais comumente encontradas em ambientes de desenvolvimento segundo a OWASP. É apresentada também uma visão geral sobre DEVOPS. A aplicação do DevSecOps requer a manutenção das melhorias promovidas pelo DevOps. Assim, é importante explorar os seus objetivos. Finalmente, são explicados os conceitos e objetivos do DEVSECOPS, além das CONSIDERAÇÕES FINAIS DO CAPÍTULO.

## 2.2 Segurança de Software

A segurança é um conceito amplo, associado à proteção de ativos, dados e informações, sendo ainda relacionada à elaboração de políticas que detalham regras de acesso a recursos (VIEGA; MCGRAW, 2006). A Segurança de Software consiste na ideia de que o software deva permanecer funcionando corretamente após um ataque malicioso (MCGRAW, 2004).

Usuários não autorizados com acesso a uma aplicação bem como ataques de negação de serviço são consideradas violações de segurança (VIEGA; MCGRAW, 2006). Uma parcela substancial dos produtos de software em uso na atualidade não é concebida com ênfase significativa em segurança. Isso se deve ao fato de que as partes interessadas no produto não tratam a segurança do software com alta prioridade (RANSOME; MISRA, 2018). Assim, pessoas mal intencionadas têm obtido sucesso ao invadir canais válidos de autenticação por meio de técnicas como *Cross-Site Scripting* (XSS), *SQL Injection* e exploração de *buffer overflow* (RANSOME; MISRA, 2018). Tais técnicas serão mais bem explicadas ao longo do capítulo.

Em cenários nos quais produtos de software vulneráveis são implantados em ambientes de produção, também surgem outros tipos de ataques. O CERT BR (Centro de Estudos, Resposta e Tratamento de Incidentes de Segurança no Brasil) (2023) categoriza por exemplo, o DoS (*Denial of Service*), no qual o atacante usa um conjunto de computadores para tornar um serviço, dispositivo ou rede indisponível. O *Phishing*, por sua vez, é uma subcategoria de fraude, na qual os atacantes utilizam páginas e *links* falsos

visando obter vantagens, dados e credenciais. Outra subcategoria da fraude é o *Malware*, que envolve o envio de códigos maliciosos para furtar informações e credenciais. A Invasão refere-se ao ataque no qual se consegue acesso não autorizado a um computador ou rede. O *Scan* engloba varreduras em redes de computadores (*scans*), ataques de força bruta e tentativas sem êxito de exploração de falhas de segurança. A categoria Web trata dos ataque em que são comprometidos servidores web ou desfigurações de páginas na Internet. Há ainda outros ataques que não se enquadram nas demais categorias (CERT.BR, 2023).

A ideia, de mencionar alguns cenários que descrevem situações de problema em termos de segurança, é esclarecer sobre a relevância de um olhar mais criterioso a esse critério de qualidade, por vezes, negligenciado no desenvolvimento de software.

O Instituto Nacional de Padrões e Tecnologia (NIST), em seu *Computer Security Handbook*, ressalta que, para assegurar a segurança de sistemas de computadores, é essencial a implementação de medidas de proteção que garanta integridade, disponibilidade e confidencialidade dos recursos em sistemas de informação automatizados (GUTTMAN; ROBACK, 1995). A tríade da segurança da informação, conhecida como CIA (Confiabilidade, Integridade e Disponibilidade), representa os três princípios básicos da área e integra os objetivos fundamentais de segurança (STALLINGS, 2017):

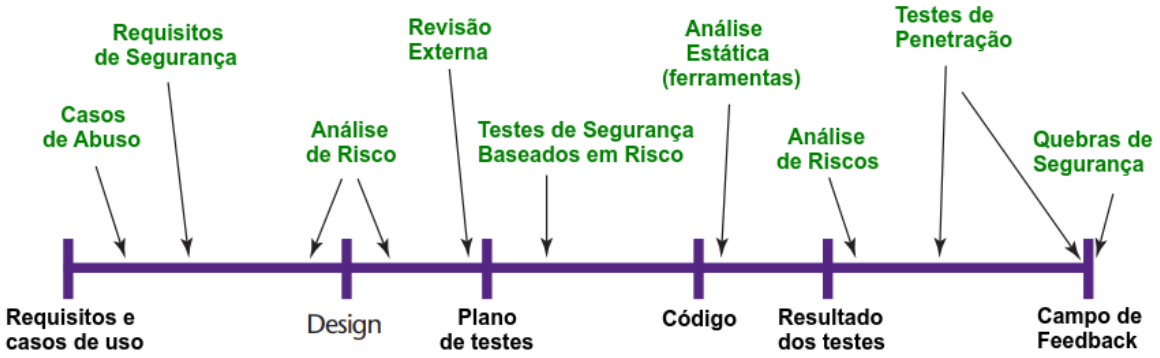
- **Confidencialidade**: a confidencialidade refere-se à preservação de restrições, acesso e divulgação de informações incluindo meios de proteger a privacidade pessoal e informações proprietárias;
- **Integridade**: a integridade está relacionada à proteção contra modificação ou destruição das informações, e
- **Disponibilidade**: a disponibilidade visa garantir acesso e uso confiáveis e ininterruptos às informações.

A Segurança de software está intimamente ligada à qualidade de um software (STALLINGS, 2017). Para assegurar confiabilidade, integridade e disponibilidade de um software, é fundamental que sejam incorporadas medidas de segurança ao longo de todo o ciclo de desenvolvimento, desde a fase inicial de *design*, desenvolvimento, testes, implementação, até manutenção contínua. Garantir que as diretrizes de segurança sejam parte integrante do processo de desenvolvimento ajuda a prevenir vulnerabilidades, proteger informações sensíveis e manter a disponibilidade dos sistemas, contribuindo assim para a construção de software seguro e confiável (STALLINGS, 2017).

Um software seguro mantém sua corretude e previsibilidade resistindo aos esforços intencionais para comprometer sua confiabilidade (GOERTZEL et al., 2007). Para garantir um software seguro, existe um conjunto de boas práticas que devem ser seguidas

durante o ciclo de vida do desenvolvimento de software, incluindo a integração de diretrizes de segurança. A Figura 1 ilustra o ciclo de vida de desenvolvimento de software, destacando as etapas correspondentes à segurança.

Figura 1 – Ciclo de vida de software com etapas relacionadas à segurança




Fonte: (MCGRAW, 2004) (Tradução: Autores)

- **Requisitos de segurança**: uma boa prática é a incorporação do levantamento de requisitos de segurança entre as fases de levantamento de requisitos e *design* do software. Este levantamento refere-se à segurança evidente e, de forma análoga, à elaboração dos casos de uso. Esses casos de uso propõem a formulação de cenários de abuso, que descrevem o comportamento do sistema durante uma situação de ataque, contribuindo para uma abordagem mais preparada na garantia da segurança;
- ***Design* e arquitetura**: a incorporação de medidas de segurança durante a fase de *Design* e Arquitetura visa à concepção de um sistema coerente e dotado de arquitetura unificada, considerando a aplicação de princípios de segurança e análises de riscos de ataques de forma criteriosa;
- **Implementação de código**: durante a fase de implementação, é necessária a correção de falhas e, principalmente, vulnerabilidades detectadas por meio de ferramentas de análise estática;
- **Testes de segurança**: a fase de testes de segurança aborda a construção de planos de teste que incluem estratégias pars testar funcionalidades de segurança com ajuste padrão, ou seja, com base em padrões de ataque e modelos de ameaças;
- **Testes de penetração**: na sequência do ciclo de vida de desenvolvimento de software, esta fase encontra-se mais ao final, uma vez que se refere à aplicação de testes com a representação de um atacante externo, visando à compreensão do funcionamento do software em seu ambiente de produção com o objetivo de identificar vulnerabilidades e corrigí-las. Os testes de penetração devem levar em consideração a arquitetura do sistema, a fim de evitar uma avaliação superficial, e

- **Monitoramento do software**: a equipe de operação deve ser responsável por monitorar o sistema em funcionamento, com o propósito de identificar vulnerabilidades e quebras de segurança que podem ocorrer durante a operação do software.

Desenvolver código seguro demanda atenção de todos os elementos relacionados à execução de um programa, incluindo o ambiente no qual ele opera e a natureza dos dados que manipula (STALLINGS, 2017). Dessa forma, em 2008, a Microsoft propôs um detalhamento maior de práticas que incorporam a segurança no ciclo de vida de software (Microsoft, 2016), tais como: providenciar treinamento; definir requisitos de segurança; definir métricas e relatório de conformidade; modelar ameaças; estabelecer requisitos de *design*; definir e usar padrões de criptografia; gerenciar o risco de segurança de utilizar componentes de terceiros; usar ferramentas aprovadas; realizar testes de segurança de análise estática (SAST); realizar testes de segurança de análise dinâmica (DAST); realizar testes de penetração, e estabelecer um padrão de processo de resposta de incidentes. No entanto, além da atenção constante ao longo de todo o ciclo de vida de software, é fundamental a realização de testes de segurança para garantir o funcionamento do sistema como é esperado (WOTAWA, 2016).

### 2.2.1 Testes de Segurança no Ciclo de Vida de Desenvolvimento de Software

A segurança de software está relacionada a *bugs* encontrados por pessoas mal intencionadas, nos quais a entrada difere do que é esperado. Portanto, tais *bugs* não são encontrados facilmente em testes comuns (STALLINGS, 2017). Por isso, deve ser feita uma série de testes de segurança. Três ferramentas primárias são básicas para esta análise e são categorizadas como *"fuzzing"*, análise estática e análise dinâmica, conforme acordado em Ransome; Misra (2018):

- *Fuzzing*: com *design* de teste simples, o teste *fuzz* é um teste caixa preta automatizado ou semi automatizado que consiste no envio de dados de entrada inesperados, inválidos ou malformados para um software;
- Análise Estática: conhecida como testes de segurança de análise estática (SAST), é a análise realizada em código fonte sem a necessidade da execução deste. Sendo assim, são análises realizadas por ferramentas automatizadas que identificam vulnerabilidades na fase de implementação e sua detecção ocorre no nível da linha de código, permitindo correções eficientes destas vulnerabilidades, e
- Análise Dinâmica: conhecida como testes de segurança de análise dinâmica (DAST), é a análise do software em execução, e seu objetivo é depurar o programa para todos os cenários para os quais foi projetado, realizando ataques em tempo real.

A inclusão de etapas de segurança no ciclo de vida de um software e a realização de testes de segurança, quando alinhados com as diretrizes OWASP, elevam a segurança do software significativamente (BJöRNHOLM, 2020).

Outro teste fundamental para assegurar a segurança e o comportamento adequado do software são os Testes de Sistema ou Testes de Interface, que reproduzem o comportamento de um usuário final ao interagir com o sistema (VALENTE, 2020). Esses testes ajudam a garantir que as funcionalidades do software atendam aos requisitos de segurança, levando em conta os casos extremos e/ou os comportamentos inesperados.

## 2.3 OWASP

Criada por Mark Curphey em Setembro de 2001 (HUSEBY, 2004), o OWASP (em inglês, *Open Web Application Security Project*), ou Projeto Aberto de Segurança em Aplicações Web, é uma fundação sem fins lucrativos. Essa fundação tem como objetivo divulgar e promover boas práticas de segurança no desenvolvimento web através da disponibilização gratuita de ferramentas, documentos e pesquisas, todos relacionados com segurança de software, visando conscientizar os desenvolvedores dos cuidados a serem tomados na construção de aplicações web (OWASP Foundation, 2001a).

"WebScarab" e "WebGoat" são exemplos de aplicações fornecidas gratuitamente pela OWASP. O primeiro consiste em uma ferramenta de alta qualidade, gratuita, para a verificação de vulnerabilidades, enquanto que o segundo é um ambiente interativo para aprendizado de vulnerabilidades de aplicações web (HUSEBY, 2004). Além da disponibilização das mais variadas aplicações, a comunidade OWASP também disponibiliza diversas pesquisas e diferentes estudos relacionados ao tema de segurança. O OWASP compreende documentos que estão entre os mais visados e referenciados sobre os cuidados a serem tomados no processo de desenvolvimento ao considerar segurança de software como um critério de relevância. Dentre esses documentos, consta uma lista contendo as dez vulnerabilidades mais relevantes no desenvolvimento de aplicações web (originalmente *The Ten Most Critical Web Application Security Vulnerabilities*), também conhecida como OWASP Top Ten (OWASP Foundation, 2003).

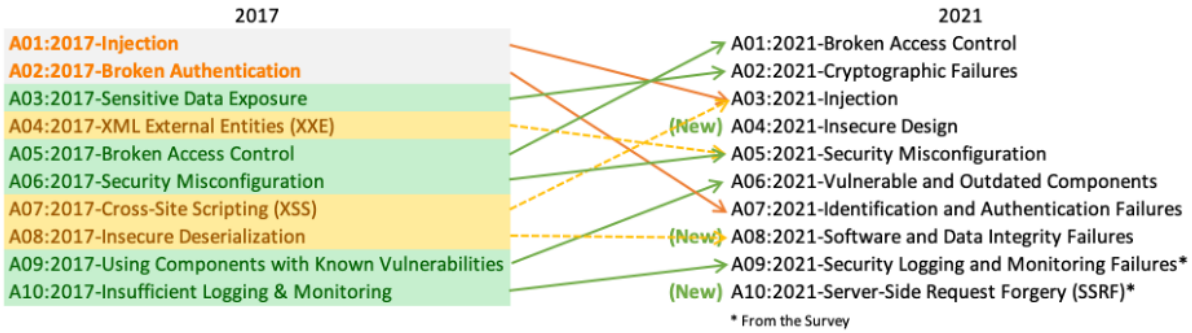
### 2.3.1 OWASP Top Ten

Disponibilizada inicialmente em 2003, o "OWASP Top Ten" consiste em um conjunto de dez vulnerabilidades e ameaças de maior relevância no desenvolvimento web. Esse conhecimento é de fundamental importância, já que as vulnerabilidades relatadas são justamente as mais usadas por pessoas mal intencionadas. Além de disponilizar a descrição e o funcionamento de tais vulnerabilidades, o OWASP também fornece medidas

de prevenção e cuidados a serem tomados, ajudando os desenvolvedores a adotar medidas proativas para mitigar os riscos associados a essas ameaças (OWASP Foundation, 2003).

A cada três a quatro anos, a comunidade do OWASP disponibiliza uma versão atualizada da lista de vulnerabilidades. Essa atualização ocorre com base em análises e em pesquisas relacionadas com o aumento de ocorrências e exploração de uma determinada vulnerabilidade por pessoas mal intencionadas, onde novas vulnerabilidades podem ser adicionadas, enquanto outras podem ser retiradas. Adicionalmente, é comum que os itens mudem de posições com base no aumento de risco associado à vulnerabilidade. Nesse sentido, a ordem dos itens é relevante. No momento da escrita desta monografia, a última versão disponilizada pelo OWASP foi referente ao ano de 2021 (OWASP Foundation, 2003). A Figura 2 apresenta as mudanças de posição das vulnerabilidades, listando as Top 10 vulnerabilidades em 2021.

Figura 2 – Mudanças ocorridas de 2017 para 2021




Fonte: OWASP*

A seguir, estão listadas todas as dez vulnerabilidades do OWASP Top Ten 2021, com uma breve explicação sobre cada uma no que compreende seu funcionamento e como se proteger dela. Cabe ressaltar que os autores procuram traduzir cada vulnerabilidade, para o português, considerando não perder semântica com o idioma em inglês.

- **A01 - Quebra de Controle de Acesso (A01:2021-*Broken Access Control*):**

  Na última edição do documento, saindo da 5ª para a 1ª posição no *ranking*, a vulnerabilidade conhecida como Quebra de Acesso foi identificada em várias aplicações nas pesquisas feitas pela OWASP.

  Podendo ser explorada de várias formas, essa vulnerabilidade, em geral, ocorre na comunicação com o cliente, isto é, na comunicação do usuário com o servidor da

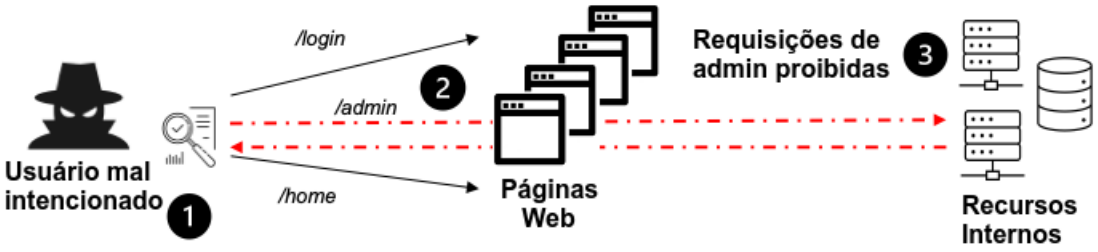
*OBS: Os autores mantiveram a imagem sem tradução para conferir a semântica exata de cada termo citado, não havendo correspondência clara para alguns termos, quando colocados em português.

aplicação. Em uma aplicação vulnerável, alguém malicioso consegue realizar a troca de parâmetros, como IDs, visando o acesso a páginas e dados de acesso restrito aos administradores (OWASP Foundation, 2021a).

A vulnerabilidade de Quebra de Acesso é possível devido à ausência de verificação e controle dos privilégios do usuário ao carregar determinadas páginas da aplicação, podendo ser mitigada com a utilização de perfis para os usuários, delimitando o acesso às páginas, dependendo do perfil de cada tipo de conta. Esta vulnerabilidade também pode ser explorada a partir de requisições na API e na configuração do CORS da aplicação. Sendo assim, cuidados na configuração são necessários (OWASP Foundation, 2021a).

A Figura 3 explifica a aplicação do ataque de Quebra de Acesso.

Figura 3 – Exemplo de ataque por quebra de controle de acesso




Fonte: MyF5 (2023) (Tradução: Autores)

Através de uma ferramenta, a pessoa mal intencionada realiza uma busca por páginas que estejam sendo protegidas. Na Figura 3, essa pessoa descobre que a página de /admin da aplicação não está sendo restringida e, a partir desse acesso, realiza ações que não deveriam estar disponíveis para ele.

- **A02 - Falhas de Criptografia (A02:2021-*Cryptographic Failures*):**

  Subindo uma posição em relação à pesquisa realizada em 2017, esta vulnerabilidade recebeu uma atualização no seu nome: até a pesquisa anterior, a vulnerabilidade era conhecida como "Exposição de Dados Sensíveis" (*Sensitive Data Exposure*). Essa mudança ocorreu devido ao desejo por parte dos pesquisadores de trazer uma maior enfâse no aspecto da criptografia dos dados em si (OWASP Foundation, 2021b).

  A categoria trata da aplicação de algoritmos de criptografia fracos, ou até mesmo da ausência de criptografia, no tratamento dos dados sensíveis dos usuários cadastrados na aplicação. Essa negligência permite que, no caso de um vazamento de dados, a obtenção de senhas e outros dados ocorra de forma muito mais fácil por parte de uma pessoa mal intencionada. Para aplicações que usam algoritmos de criptografia vulneráveis, basta essa pessoa utilizar-se das vulnerabilidades para obter dados

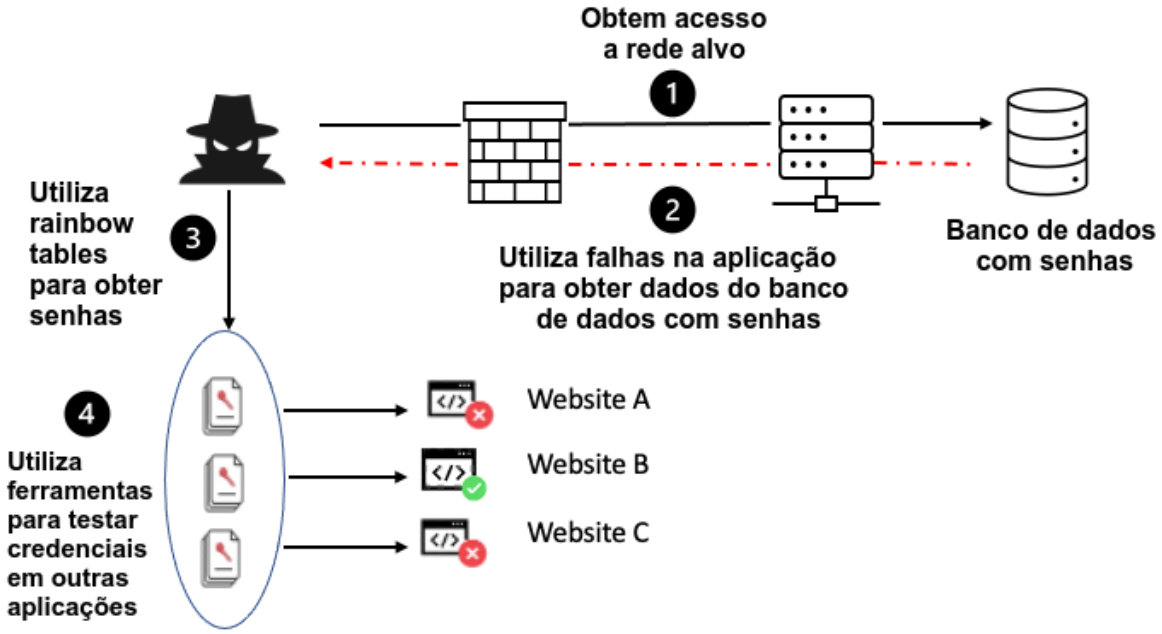
sensíveis. Já no caso da ausência completa no uso de criptografia, a pessoa mal intencionada já tem acesso direto a esses dados (OWASP Foundation, 2021b).

Um vetor de ataque é a não utilização de protocolos de comunicação seguros, como o HTTPS, em aplicações que tratam de dados sensíveis, facilitando o acesso por terceiros aos dados dos usuários durante o processo de transferência entre cliente e servidor (OWASP Foundation, 2021b).

A prevenção está ligada à utilização de algoritmos seguros bem como de mecanismos que reforcem conexão criptografada do usuário com a aplicação (OWASP Foundation, 2021b).

A Figura 4 demonstra a vulnerabilidade de falhas de criptografia.

Figura 4 – Exemplo de ataque de falha de criptografia




Fonte: MyF5 (2023) (Tradução: Autores)

A Figura 4 mostra a utilização de alguma outra vulnerabilidade, visando acesso a registros do banco de dados da aplicação e, com a utilização de *rainbow tables*, a pessoa mal intencionada consegue obter a senha do usuário. A partir desse momento, é feito um processo de análise em outros sites que possam utilizar a mesma senha.

- **A03 - Injeção (A03:2021–*Injection*):**

  Ocupando anteriormente a 1ª posição na pesquisa realizada em 2017, passando agora para a 3ª posição, ataques de Injeção não são novos na indústria.

  O mais conhecido dos ataques de Injeção é o de Injeção SQL. O conceito desse tipo de ataque tem origem nos anos 90. Em 1998, Rain Forest Puppy publicou um artigo na Phrack Magazine descrevendo a lógica e o funcionamento do ataque, que,

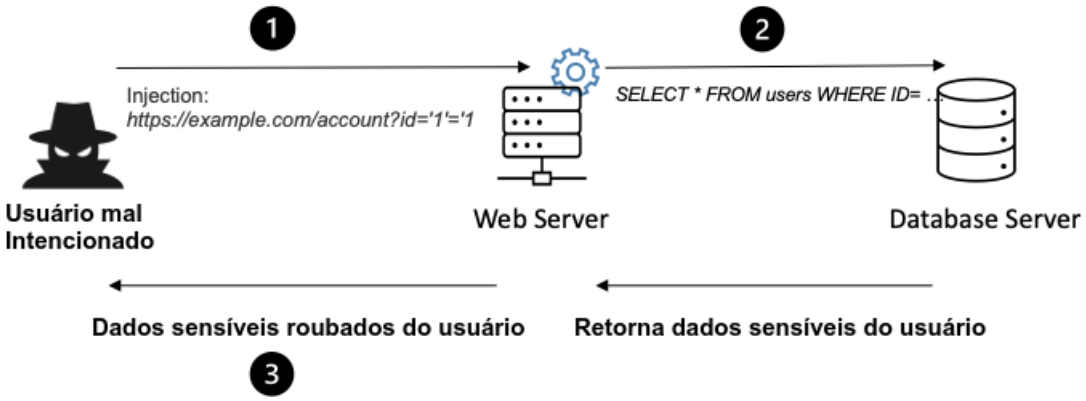
até hoje, tem alta relevância no aspecto da segurança de banco de dados (LAN; ZHANG, 2017).

O ataque de Injeção consiste na inserção, por parte da pessoa mal intencionada, de uma instrução não esperada que, ao ser processada na aplicação, retorna um resultado não desejado pelos desenvolvedores, mas sim por essa pessoa, permitindo o acesso a dados sensíveis de outros usuários. Com esse tipo de ataque, é possível obter praticamente qualquer informação no banco de dados da aplicação, sendo os mais almejados os dados de acesso a contas de administradores, como nomes de usuário, email e senha. Dependendo dos dados armazenados pela aplicação, ataques de injeção também podem ser usados para obter dados pessoais dos usuários (OWASP Foundation, 2021c). Esses dados são muito visados por pessoas mal intencionadas, uma vez que podem ser vendidos em massa na DarkWeb (OUELLET et al., 2022).

A prevenção pode ser feita de várias formas, focando parametrização dos dados enviados pelo usuário, além do tratamento e da sanitização dos dados enviados (OWASP Foundation, 2023b). Ferramentas de análise estática de código também são muito utilizadas para identificar vetores de ataque (LAN; ZHANG, 2017).

A Figura 5 apresenta a vulnerabilidade de injeção.

Figura 5 – Exemplo de ataque de injeção




Fonte: MyF5 (2023) (Tradução: Autores)

Na Figura 5, a pessoa mal intencionada insere comandos visando obter acesso a dados na aplicação. Esses comandos, não sendo sanitizados, podem permitir à pessoa acessos não autorizados; negação de serviço (DoS), e obtenção de dados sensíveis.

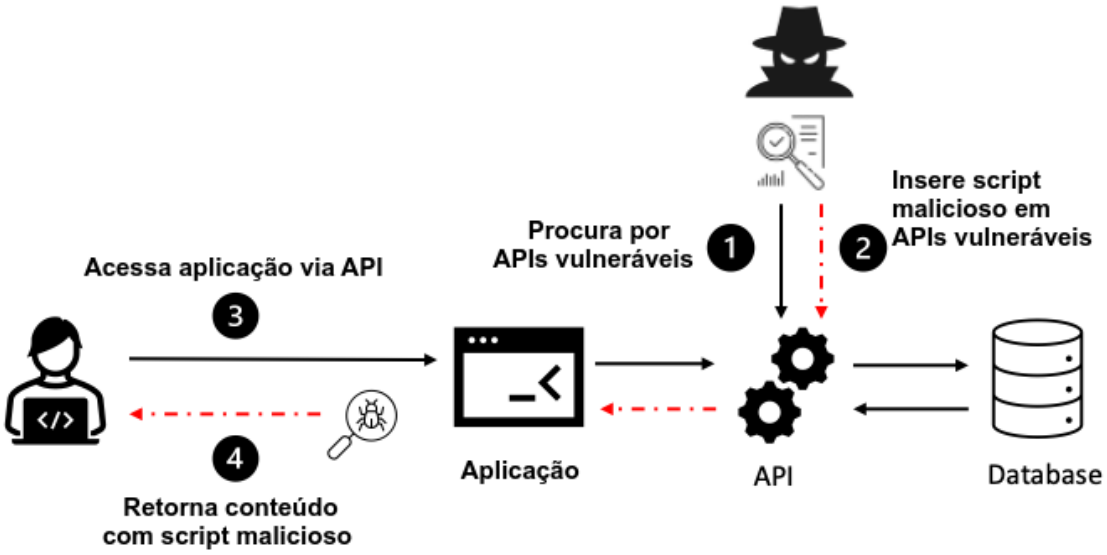
- **A04 - *Design* Inseguro (A04:2021-*Insecure Design*):**

  Introduzida em 2021, essa categoria trata de falhas de *design* e arquitetura na implementação da solução, além da ausência de um modelo de ameaças e de padrões de segurança (OWASP Foundation, 2021d).

A OWASP realiza uma distinção entre um *design* inseguro e uma implementação insegura: basicamente, uma aplicação com um *design* inseguro não pode ser protegida nem mesmo com uma implementação perfeita.

A Figura 6 exemplifica a vulnerabilidade de *design* inseguro em uma situação em que um atacante consegue ter acesso a uma API vulnerável.

Figura 6 – Exemplo de ataque realizado a partir de um *design* inseguro




Fonte: MyF5 (2023) (Tradução: Autores)

Na Figura 6, a pessoa mal intencionada consegue inserir *scripts*. Eventualmente um outro usuário ira requisitar um dado da API que, então, poderá retornar o *script* malicioso para o usuário.

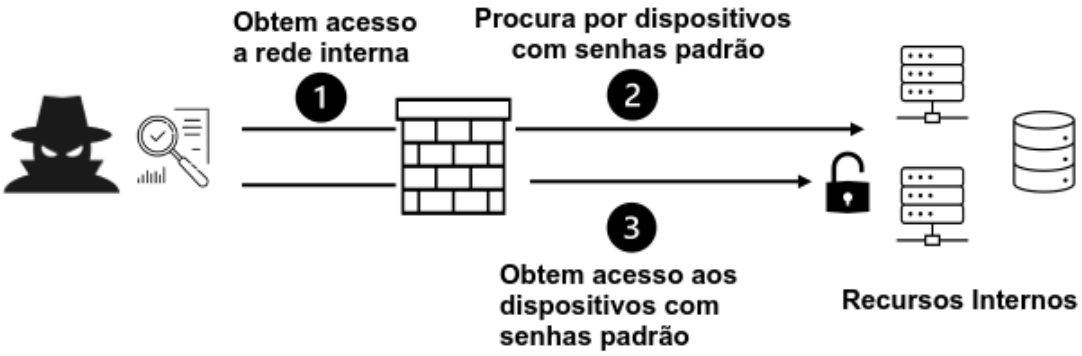
- **A05 - Configuração Incorreta de Segurança (A05:2021-*Security Misconfiguration*):**

  Ocupando a 6ª posição em 2017, essa categoria foi unificada com a categoria *XML External Entities*, que existia até o ano de 2017, quando ocupou a 4ª posição. Após a unificação e com um aumento na complexidade das ferramentas utilizadas, essa categoria passou a ocupar a quinta posição em 2021 (OWASP Foundation, 2021e).

  Como exemplo de aplicações vulneráveis, têm-se aquelas que utilizam configurações padrão. Essas configurações são projetadas para um ambiente de desenvolvimento. Portanto, a utilização delas em um ambiente para uso do usuário comum representa um risco, já que existe um menor controle de acesso. Outro aspecto abordado pela A05 é a ativação de funcionalidades não utilizadas.

  A Figura 7 apresenta a vulnerabilidade de Configuração Incorreta de Segurança.

Figura 7 – Exemplo de ataque realizado a partir de uma configuração errada




Fonte: MyF5 (2023) (Tradução: Autores)

Na Figura 7, a pessoa mal intencionada acessa uma rede interna e procura por credenciais padrão, acessando os recursos internos.

- **A06 - Componentes Vulneráveis e Desatualizados (A06:2021- *Vulnerable and Outdated Components*):**

  Problema conhecido e caracterizado por desafios na realização de testes e na avaliação dos riscos, subiu da 9ª posição em 2017. É a única categoria que não possui *Common Vulnerability and Exposures* (CVEs) mapeadas. Portanto, são atribuídos pesos de 5.0 e uma exploração padrão para sua pontuação (OWASP Foundation, 2021f).

  A vulnerabilidade é explorada a partir de ferramentas e componentes desenvolvidos por terceiros, não estando assim sob a supervisão e o controle direto dos desenvolvedores. Tais componentes são comumente utilizados visando facilitar o desenvolvimento de uma certa funcionalidade, acelerando o processo de desenvolvimento. Esses componentes externos estão sujeitos a vulnerabilidades, que podem, ou não, serem resolvidas pelos seus originais desenvolvedores.

  A OWASP chama atenção sobre a necessidade de se manter tais componentes atualizados; conhecer as versões dos componentes utilizadas; verificar se ainda há suporte às versões utilizadas; testar a compatibilidade das bibliotecas atualizadas, e proteger as configurações dos componentes.

  A Figura 8 demonstra um tipo de ataque realizado utilizando sistemas vulneráveis.

Figura 8 – Exemplo de ataque realizado em aplicação utilizando sistemas vulneráveis


Fonte: MyF5 (2023) (Tradução: Autores)

Na Figura 8, a pessoa mal intencionada utiliza ferramentas para encontrar sistemas desatualizados, além de explorar as *flags* encontradas adicionando códigos maliciosos.

- **A07 - Falhas de Identificação e Autenticação (A07:2021-*Identification and Authentication Failures*):**

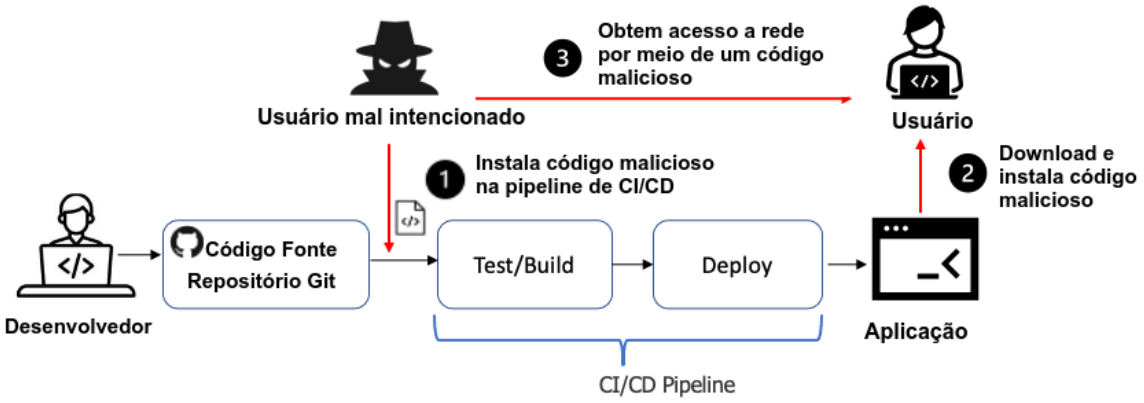
  Antes conhecida como Falhas de Autenticação, agora, inclui mais CWEs relacionadas a falhas de identificação. Apesar de ainda estar nas Top 10 vulnerabilidades, a disponibilidade de estruturas padronizadas está auxiliando para que esta vulnerabilidade não apareça tanto (OWASP Foundation, 2021g).

  Confirmar identificação, autenticação e gerenciamento de sessão é fundamental para se proteger contra esta vulnerabilidade. Aplicações que permitem ataques de força bruta; senhas fracas e padronizadas; recuperação de credenciais com respostas conhecidas; exposição de id da sessão na url; não invalidação de identificadores de sessão, e reutilização após *login* com sucesso são exemplos de aplicações com a vulnerabilidade de falhas de identificação e autenticação. Lembrando que reutilização de serviços ou outros recursos, dependendo da situação, deveria demandar novo *login*, para garantir maior segurança. Há aplicativos, por exemplo, que realizam checagem dupla, com validações extras via celular, por exemplo, além da senha digitada pelo usuário.

  Para prevenir esta vulnerabilidade, há algumas práticas para adotar, como autenticação por múltiplos fatores; realização de *deploy* sem utilizar credencial padrão; implementação de verificação de senhas fracas; limitação de muitas tentativas de *login* sem sucesso, e utilização de gerenciador de sessão seguro.

  A Figura 9 apresenta um exemplo de ataque utilizando falhas de identificação e autenticação.

Figura 9 – Exemplo de ataque realizado com falhas no processo de identificação e autenticação




Fonte: MyF5 (2023) (Tradução: Autores)

Na Figura 9, a pessoa mal intencionada utiliza a falha de identificação e autenticação no repositório do github do desenvolvedor, adicionando código malicioso no *pipeline* *Continuous Integration/Continuous Delivery* (CI/CD), conseguindo acesso à rede do usuário.

- **A08 - Falhas de Integridade de Software e Dados (A08:2021-*Software and Data Integrity Failures*):**

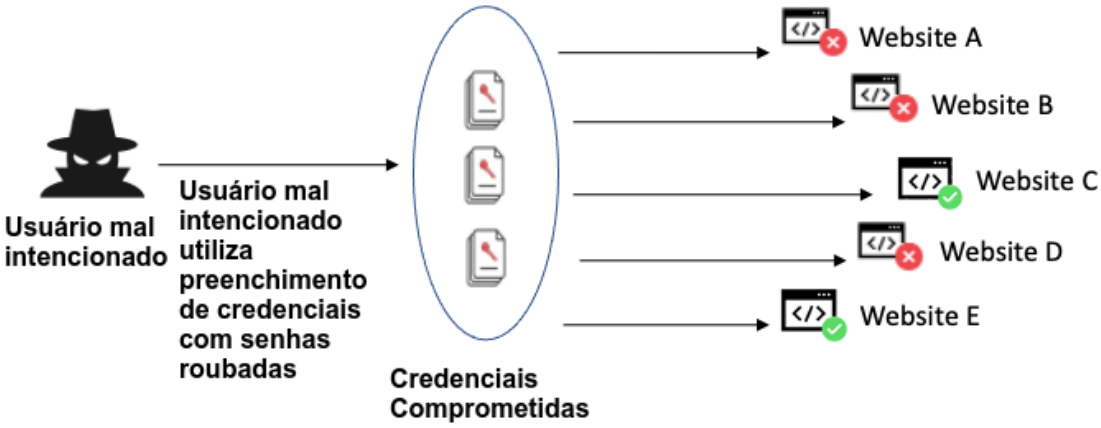
  Categoria surgida em 2021, focada em atualizações de software, dados críticos e *pipelines* CI/CD que não verificam integridade. Tem um dos maiores impactos nos CVEs e CVSS, já que são mapeadas 10 CWEs sobre este tópico (OWASP Foundation, 2021h).

  São relacionadas a código e infraestrutura frágeis para ataques de integridade. Um *pipeline* CI/CD inseguro, por exemplo, tem alta probabilidade de acesso não autorizado, código malicioso e comprometimento do sistema. Atualmente, muitas aplicações possuem atualizações automáticas e pouca verificação de integridade, fazendo com que uma pessoa mal intencionada carregue suas próprias atualizações para distribuir.

  Algumas maneiras de prevenção é a utilização de assinatura digital; bibliotecas consumidas de repositórios confiáveis; ferramentas que verificam vulnerabilidades em componentes; revisão constante de código e configuração; garantia de que o *pipeline* de CI/CD tenha segregação; configuração e controle de acesso adequados para a integridade do código que flui através dos processos de construção e implantação, e garantia de que dados serializados, não assinados e não criptografados não sejam enviados para clientes não confiáveis.

  A Figura 10 demonstra um ataque de falhas de integridade de software e dados.

Figura 10 – Ataque realizado por falhas de integridade do software




Fonte: MyF5 (2023) (Tradução: Autores)

Na Figura 10, a pessoa mal intencionada preenche credenciais com senhas roubadas, e acessa uma série de sistemas web.

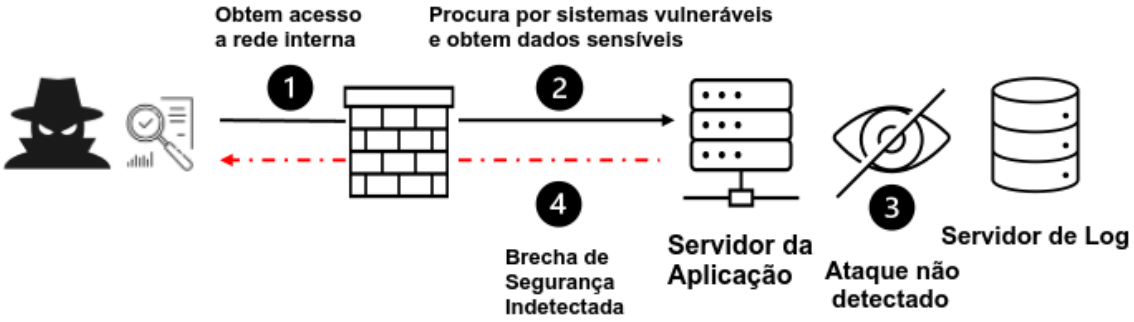
- **A09 - Falhas de Registro e Monitoramento de Segurança (A09:2021-*Security Logging and Monitoring Failures*):**

  Categoria difícil de testar e não muito bem representada nos CVEs e CVSS, mas as falhas podem impactar análise forense, visibilidade e alertas de incidentes (OWASP Foundation, 2021i). Entende-se por análise forense, aquela que permite o uso de método científico, envolvedo até mesmo variados domínios, considerando viés jurídico/criminal. Esta categoria objetiva detectar, escalar e responder violações de segurança. Muitas vezes acontecem alguns cenários de falhas de registro e monitoramento de segurança, tais como:

  – Eventos auditáveis como *login* e falhas de *login* não são registrados;
  – Erros e avisos geram *logs* pouco descritivos;
  – *Logs* são armazenados somente localmente;
  – *Logs* de API não são monitorados visando buscar atividades suspeitas;
  – Testes de penetração e ferramentas DAST não geram alertas, e
  – A aplicação não detecta invasões em tempo real.

  A Figura 11 demonstra um ataque realizado em um sistema que não possui um sistema de *logs*.

Figura 11 – Exemplo de ataque realizado por falhas de registro e monitoramento do segurança




Fonte: MyF5 (2023) (Tradução: Autores)

Na Figura 11, a pessoa mal intencionada consegue acesso a rede e obtém dados sensíveis. A ausência de um sistema de *logs* que permita o rastreamento e registro de operações e acessos ao sistema promove uma perpetuação do ataque.

- **A10 - Falsificação de Solicitação do Lado do Servidor (A10:2021-*Server-Side Request Forgery*):**

  Categoria adicionada a partir da primeira posição da pesquisa da comunidade Top 10. Embora os dados indiquem uma baixa frequência de ocorrência nessa categoria, a comunidade enfatiza sua importância como uma vulnerabilidade significativa (OWASP Foundation, 2021j).

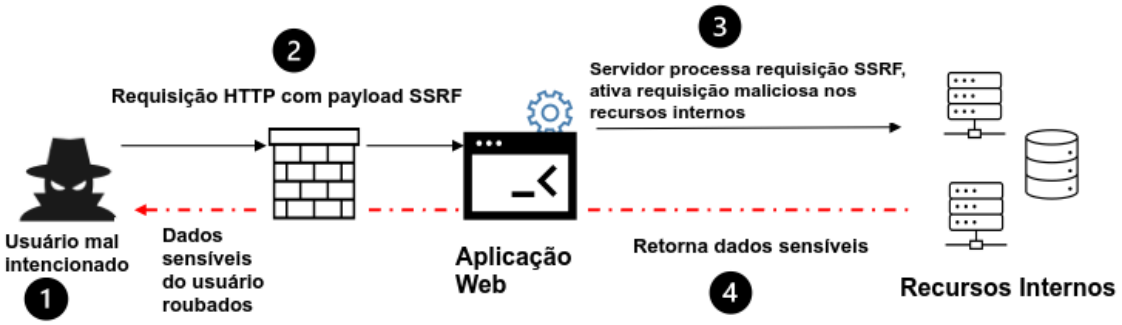
  O SSRF (*Server Side Request Forgery*) ocorre quando uma aplicação permite que o invasor influencie nas solicitações para outros sistemas, e principalmente quando a aplicação busca um recurso remoto, sem validar a URL fornecida pelo usuário. Nesta categoria, um invasor consegue fazer solicitações não autorizadas a partir do servidor, permitindo solicitação para um destino inesperado, mesmo quando protegido por *firewall*, *Virtual Private Networks* (popularmente conhecido como VPN, consiste na criação de uma comunicação intermediária entre redes) ou *Access-Control List* (lista que define as permissões que cada usuário e processo executado no sistema deve ter acesso) de redes.

  Alguns cenários de ataque SSRF são determinação de portas abertas em servidores internos; exposição de dados confidenciais e arquivos locais como *file:///etc/passwd* e *http://localhost:28017/*; acesso ao armazenamento de metadados em serviços em nuvem, e comprometimento na disponibilidade de serviços como *Remote Code Execution* (RCE) e *Denial of Service* (DoS). Em um ataque RCE, o invasor possui acesso completo a uma máquina de forma remota (ScienceDirect, 2023b). Já um

ataque DoS tem como objetivo derrubar um serviço através de uma inundação de requisições em um curto espaço de tempo (ScienceDirect, 2023a).

A Figura 12 apresenta um exemplo de ataque com SSRF.

Figura 12 – Exemplo de ataque de falsificação de solicitação do lado do servidor




Fonte: MyF5 (2023) (Tradução: Autores)

Na Figura 12, o servidor processa uma requisição SSRF com *triggers* maliciosos que permitem acesso aos recursos internos.

## 2.4 DevOps

O termo DevOps foi originado em 2008 por Patrick Debois e Andrew Shafer. Passou a ser usado comumente em 2009, após a *Velocity Conference* com a apresentação *10+ Deploys Per Day: Dev and Ops Cooperation at Flickr* feita por John Allspaw e Paul Hammond (KIM; BEHR; SPAFFORD, 2018). O resultado técnico, cultural e arquitetural da abordagem DevOps depende de muitos movimentos gerenciais e filosóficos, como *Lean*, Teoria das Restrições, metodologias ágeis, Sistema de Produção Toyota, cultura segura, fatores humanos e outros (KIM et al., 2021). Apenas para contextualizar um pouco, Lean é uma filosofia de gestão inspirada em práticas e resultados do Sistema Toyota (Lean Enterprise Institute, 2023). Nesse caso, busca-se desperdício zero, potencializando resultados e recursos humanos, e evitando esforços, quando algo pode ser reaproveitado/compartilhado. Nessa mesma linha, segundo Goldratt e Cox (2002), a Teoria das Restrições compreende um modelo conceitual de gestão que considera qualquer sistema gerenciável como sendo limitado por um pequeno número de restrições.

O DevOps surgiu com o propósito de unificar as equipes de segurança e operações, constituindo um conjunto de práticas cujo objetivo consiste na redução do intervalo temporal entre os *commits* realizados no sistema e a implantação em ambiente de produção, assegurando ainda um padrão elevado de qualidade (BASS; WEBER; ZHU, 2015). De acordo com Bass; Weber; Zhu (2015), a abordagem DevOps originou-se da necessidade de

agilizar a implementação e a operação de projetos de engenharia de software e hardware no departamento de tecnologia. Isso permitiu a adoção de uma cultura em que o ciclo de vida é acelerado; integra melhores práticas; foca na redução de riscos em projetos críticos; é integrado em estrutura, e resulta em um aumento de produtividade.

O DevOps foi implementado com o objetivo de fornecer uma estratégia de comunicação e colaboração que permitisse à instituição estabelecer um *feedback* baseado nas melhores práticas de gestão de projetos e gerenciamento de riscos (GóMEZ et al., 2017). Ainda segundo Gomez et al. (2017), o DevOps automatiza processos e procedimentos, considerando pessoas e tecnologias, permitindo o estabelecimento de níveis de serviço quantitativos e qualitativos; acordos de tolerância a falhas; melhoria contínua, e o ciclo de vida de gestão de projetos adotando eficientemente práticas de engenharia de software.

Em 2012, a Puppet Labs, no "Relatório sobre o estado do DevOps", analisou 4039 organizações de tecnologia, e percebeu as seguintes métricas: a implantações de código 30 vezes mais frequentes; prazo de implementação de código 8.000 vezes mais rápido; 2× a taxa de sucesso da mudança; MTTR 12x mais rápido (KIM; BEHR; SPAFFORD, 2018). MTTR (*Mean Time To Recovery* ou *Mean Time To Restore*) corresponde à média de tempo que um sistema leva para se recuperar diante de uma falha. A automação é o ponto principal do movimento DevOps e, por essa razão, devem ser publicadas atualizações pequenas e rápidas no projeto, criando uma vantagem competitiva para as empresas. Adicionalmente, o DevOps facilita a escalabilidade da implantação, e provê ambiente de teste e produção.

Para nortear o processo DevOps, há três princípios nomeados como "As três maneiras" (KIM; BEHR; SPAFFORD, 2018), sendo:

1. O primeiro princípio defende o tratamento do fluxo de trabalho da esquerda para a direita, ou seja, da equipe de desenvolvimento e operações para o cliente, lidando com pequenos lotes de trabalho sem passar os defeitos para a ramificação central. As práticas relacionadas a este princípio incluem *Build* contínuo, integração e *deploy*.

2. O segundo princípio trata-se de *feedback* rápido da direita para a esquerda, visando detecção e recuperação ágeis, e evitando reincidência dos problemas. As práticas necessárias incluem "parar a linha de produção" quando compilações e testes falham no *pipeline* de implantação, além da criação de um sistema de monitoramento e coleta de dados detalhado em ambientes de produção. Esse sistema viabiliza que todos possam ver se o código e os ambientes estão operando conforme projetado, e

3. Por fim, o terceiro e último princípio trata da criação de uma cultura que promova duas coisas: a primeira é a experimentação contínua, que exige assumir riscos e aprender com sucesso e fracasso, e a segunda é compreender que a repetição e a prática são o pré-requisito para o domínio. As práticas envolvidas neste princípio incluem alocar

pelo menos vinte por cento dos ciclos de desenvolvimento e operações de TI voltados para requisitos não funcionais, bem como reforço constante para que melhorias sejam incentivadas e comemoradas.

Apesar de seus benefícios, a implantação de uma cultura DevOps pode encontrar barreiras que podem ser de três tipos (MUñOZ; NEGRETE; MEJíA, 2019), sendo:

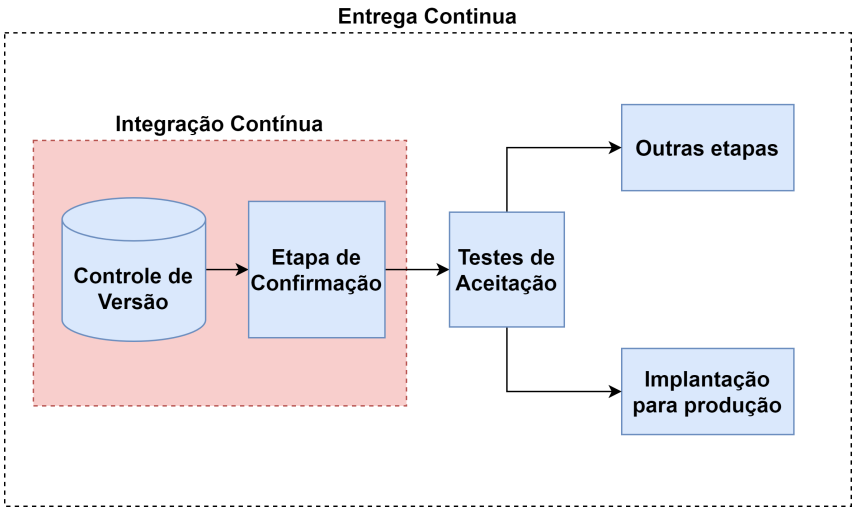
- Processos: problemas para implantar e utilizar ferramentas de automação; processos não definidos; sistemas imaturos sem controle de versão, e sistemas de software cujas linhas de processo são diferentes, como jogos e embarcados. Jogos e sistemas embarcados demandam, frequentemente, *pipelines* diferenciadas, muitas vezes fortemente dependentes de decisões em tempo real, alto desempenho, equipes multifacetadas (multi e inter disciplinares), dentre outras demandas;
- Orientação: falta de experiência em implementação de testes; falta de organização e gestão; falta de orientação, e
- Equipe: mudança de hábitos; falta de comunicação entre membros do time; receio da empresa quanto à mudança de cultura devido aos riscos; frustração do time ao tentar implementar DevOps.

No contexto do DevOps, para reduzir o tempo de diagnóstico e correção de problemas, existem algumas práticas conhecidas como *TreatOps* (Operações de Tratamento). Estas práticas são um conjunto de ações para tratar, monitorar, manter ou garantir a saúde e o desempenho contínuo de sistemas em produção. Envolve os operadores na fase de requisitos: (i) permitindo *logs* e monitoramento; (ii) tornando os desenvolvedores mais responsáveis pelos incidentes e reduzindo tempo de identificação dos erros; (iii) reforçando o processo de implantação para garantir a qualidade do *deploy* e reduzir o tempo de reparação dos erros; (iv) *deploy* contínuo e redução do tempo entre o *commit* e a implantação do código, e (v) códigos de infraestrutura e *scripts* de *deploy* para garantir alta qualidade em cada implantação e que o *deploy* prossiga como planejado (BASS; WEBER; ZHU, 2015).

O desenvolvimento de software muda rapidamente e de forma contínua (MUñOZ; NEGRETE; MEJíA, 2019). Para atender o contexto desta mudança, o DevOps aborda conceitos fundamentais como Integração Contínua e Entrega Contínua. A Figura 13 demonstra os dois processos no *pipeline* DevOps.

A Integração Contínua engloba controle de versão e o estágio do *commit*. A Entrega Contínua é uma extensão da Integração Contínua, mas com o estágio de testes de aceitação, outros estágios, e próprio *deploy* para ambiente de produção.

Figura 13 – Integração Contínua e Entrega Contínua no *pipeline* DevOps




Fonte: (LAUKKANEN; ITKONEN; LASSENIUS, 2017) (Tradução: Autores)

### 2.4.1 Integração Contínua (CI)

A Integração Contínua define um fluxo de trabalho que estabelece implementações de trechos de código em ciclos curtos, mantendo estes trechos em ramificações do código-fonte principal (VEHENT, 2018). Cada vez que os desenvolvedores realizam uma alteração, recebem os resultados dos testes automatizados. Estes testes automatizados permitem *feedback* e aprendizado rápidos, proporcionando aos desenvolvedores a oportunidade de corrigir os problemas detectados imediatamente quando os testes automatizados falham (FORSGREN; HUMBLE, 2015).

A Integração Contínua deve conter um conjunto de testes automatizados; uma cultura para interromper a linha de produção se qualquer teste falhar, e desenvolvedores realizando pequenas alterações de código (KIM et al., 2016). A ideia da Integração Contínua é que os desenvolvedores não trabalhem em códigos isolados por períodos de tempo muito grandes, mas realizem integrações frequentes e diárias, tornando o processo mais ágil. O código desenvolvido na ramificação é revisado por outro desenvolvedor. Após a revisão, este código é incorporado ao repositório central, e está pronto para a implantação (VEHENT, 2018). A Integração Contínua permite que alterações de código sejam frequentemente integradas e testadas, assegurando a qualidade e a estabilidade.

### 2.4.2 Entrega Contínua (CD)

Outro conceito conhecido no movimento DevOps é o de Entrega Contínua. A ideia da entrega contínua é a geração de *releases* de forma contínua para utilização. Isso incorre em uma automação da implantação do software para disponibilizar aos clientes mais rapidamente (VEHENT, 2018). Assim, quando uma funcionalidade, correção de *bug*

ou atualização for concluída, pode ser implantada em ambiente de produção sem muita intervenção manual. O *pipeline* CD recupera automaticamente a versão mais atual do código-fonte, empacota-o e cria uma nova infraestrutura para ele (VEHENT, 2018).

A prática da Entrega Contínua estreita a relação entre o usuário e o produto, uma vez que novas versões são disponibilizadas de maneira constante. Além disso, essa abordagem colabora com o processo de desenvolvimento de software na adaptação a mudanças e evoluções. A prática de CD reduz o risco de implantação, permite acompanhamento confiável do progresso, além de *feedback* rápido do usuário (Martin Fowler, 2013).

## 2.5 DevSecOps

DevSecOps (Desenvolvimento, Segurança e Operações) pode ser entendido, resumidamente, como a ideia de se introduzir ferramentas automatizadas de segurança em um *pipeline* originalmente de DevOps. Desta forma, o time de desenvolvimento tem a introdução de mecanismos de segurança junto aos mecanismos tradicionais proporcionados pelo DevOps. Essa estratégia procura garantir segurança do código em execução no ambiente de produção. Um código seguro não contém falhas e fraquezas que podem ser exploradas por código malicioso ou mesmo por pessoas mal intencionadas (ex. *hackers*) (GOERTZEL et al., 2007). Então, a ideia do DevSecOps é que as equipes identifiquem vulnerabilidades antes de se tornarem graves; monitorem continuamente a segurança economizando tempo e recursos, além de garantirem maior confiabilidade do software. As equipes liberam *releases* mais rapidamente, e com segurança.

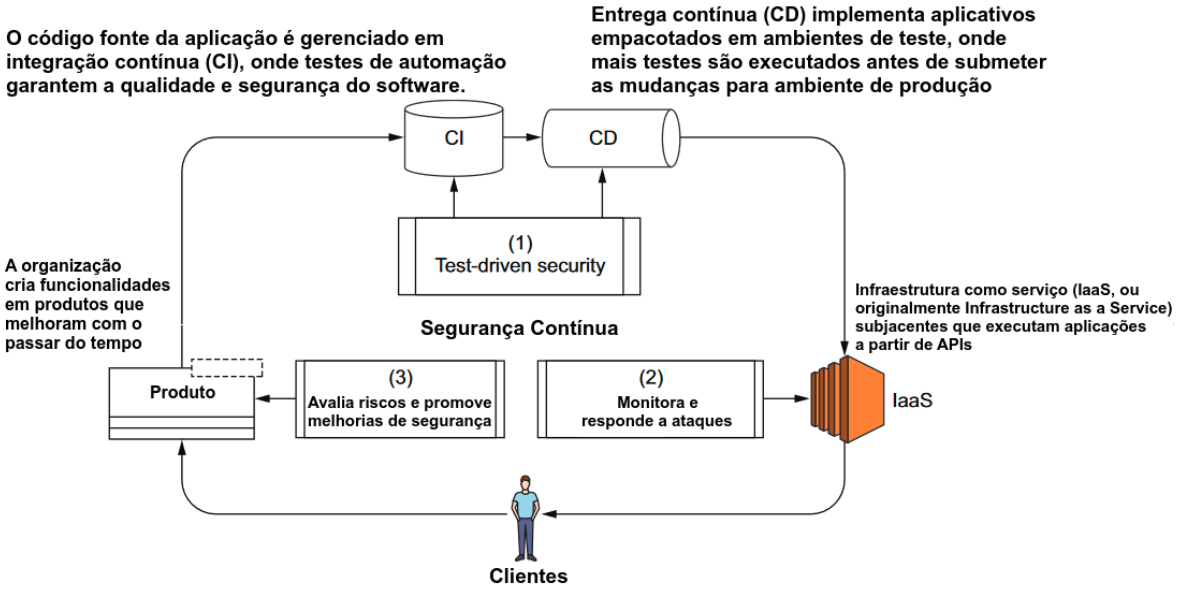
A ideia principal de acoplar segurança no DevOps é que, além de se preocuparem com a segurança das infraestruturas, a equipe de segurança também busque garantir a proteção de toda a organização, com melhoramentos contínuos (VEHENT, 2018). Por isso, o DevSecOps está diretamente relacionado à segurança contínua.

A segurança contínua é dividida em três áreas (VEHENT, 2018), sendo:

- Segurança Orientada a Testes (em inglês, *Tests Drive Security* - TDS): na maioria dos casos, os invasores de segurança procuram por alvos fáceis como estruturas da Web com vulnerabilidades de segurança; sistemas desatualizados; páginas de administração abertas com senhas adivinhadas, e credenciais de segurança vazadas. Os primeiros objetivos de segurança são aplicar controles e testar continuamente, utilizando uma lista de boas práticas formulada pela equipe;
- Monitoramento e resposta a ataques: esta área diz respeito à reação das equipes mediante um ataque de segurança, onde uma equipe de segurança deve estar preparada para agir, e

- Avaliação de riscos e amadurecimento da segurança: gerenciar riscos e testes internos e externos são estratégias fundamentais na implantação de controles de segurança, ajudando a organização a investir seus recursos de modo mais eficiente. A Figura 14 apresenta as fases da Segurança Contínua.

Figura 14 – Fases da Segurança Contínua

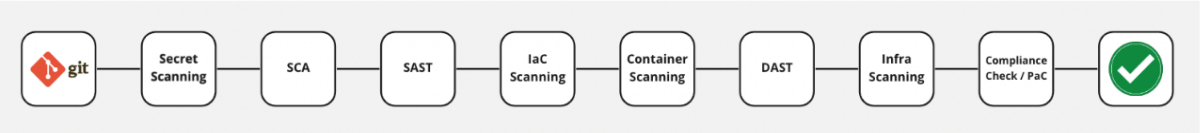



Fonte: (VEHENT, 2018)

Segundo pesquisa conduzida por Björnholm (2020), a introdução de ferramentas de segurança em um *pipeline* genérica de DevOps resultou em um aumento de 33% no tempo de *build* da aplicação. Em sua conclusão, Björnholm (2020) argumenta que o tempo extra é gerenciavel, além de mencionar que os ganhos com relação à segurança não podem ser subestimados. Assim, é possível mencionar que, segundo o estudo conduzido, o aumento no tempo de desenvolvimento constitui um dos fatores para a não utilização do DevSecOps em projetos.

A OWASP possui um *guideline* de DevSecOps que demonstra a implantação do DevSecOps e a promoção da cultura de segurança com a abordagem *Shift-Left Testing*, que significa que os testes são realizados no início dos estágios, integrando-os em todo o ciclo e não somente ao final, como em processos de desenvolvimento tradicionais (OWASP Foundation, 2023a). Isso permite o deslocamento da fase de segurança à esquerda, além da implementação da segurança continuamente. O *guideline* de DevSecOps é apresentado na Figura 15.

Figura 15 – *Guideline* DevSecOps




Fonte: OWASP*

O *pipeline* DevSecOps, ilustrado na Figura 15, apresenta as seguintes etapas:

1. *Scan*: é realizada uma verificação por vazamentos de credenciais no código-fonte, como chaves de API e para acesso ao banco de dados da aplicação;

2. Análise de Composição de Softwae (*Software Composition Analysis* - SCA): tem como objetivo realizar uma análise nas bibliotecas de terceiros utilizadas na aplicação;

3. Teste Estático de Segurança de Aplicações (*Static Application Security Testing* - SAST): consiste em um teste de caixa branca, ou seja, age realizando uma verificação do código-fonte por vulnerabilidades na aplicação, sem a execução do código;

4. IaC (*Infrastructure as a Code*) *Scanning*: compreende o gerenciamento e o provisionamento da infraestrutura por meio de código, ou seja, automatização, em vez de processos manuais, buscando identificar erros e vulnerabilidades de segurança;

5. Análise de Container (*Container Scanning*): *scan* de possível vulnerabilidades existentes em containers usados na aplicação;

6. Teste Dinâmico de Segurança de Aplicações (Dynamic Application Security Testing - DAST): se assemelha ao SAST. Entretanto, diferentemente desse último, DAST utiliza testes de caixa preta. Dessa forma, a aplicação é executada para a realização dos testes, ou seja, são testes dinâmicos e não estáticos;

7. Infra *Scanning*: visa encontrar vulnerabilidades na infra-estrutura da aplicação, evitando que tais vulnerabilidades sejam posteriormente exploradas por invasores e

8. *Compliance Check/PaC*: automatização da verificação de conformidade com as políticas definidas por meio de código, garantindo que as aplicações e infraestruturas atendam aos requisitos de segurança e conformidade exigidos.

*OBS: Os autores mantiveram a imagem sem tradução para conferir a semântica exata de cada termo citado, não havendo correspondência clara para alguns termos, quando colocados em português.

Em uma revisão da literatura disponível e conduzida por Akbar et al. (2022), visando um entendimento mais adequado sobre os desafios encontrados na migração de um modelo puramente DevOps para um DevSecOps, foi possível identificar 18 desafios, fundamentais para o DevSecOps. Cada desafio é classificado entre dez categorias. Tais desafio foram posteriormente validados a partir de uma pesquisa com especialistas. A lista de desafios encontrados consta a seguir apresentada:

- Ch1: Falta de ferramentas de teste automatizado (C1 - Teste Seguro): refere-se à ausência de ferramentas automatizadas para testar os diferentes aspectos de segurança dos processos ligados ao DevOps;
- Ch2: Negligenciar testes estáticos devido à falta de conhecimento (C1 - Teste Seguro): destaca a falta de conscientização e conhecimento dos desenvolvedores em relação à importância da utilização de testes estáticos;
- Ch3: Comunicação de padrões de segurança com DevOps (C2 - Padrões): ressalta a importância de se ter uma comunicação eficaz com a equipe de DevOps, visando a disseminação de padrões e protocolos de segurança;
- Ch4: Falta de padrões seguros de codificação (C2 - Padrões): aponta para a ausência de práticas de codificação bem definidas e padronizadas;
- Ch5: Deficiência na aplicação de políticas de segurança e na coleta de métricas (C3 - Conformidade e políticas): desafios enfrentados na concepção e na implementação de políticas de segurança consistentes, bem como na coleta de métricas que determinam sua eficácia;
- Ch6: Resistência à integração de protocolos de segurança (C3 - Conformidade e políticas): trata da resistência enfrentada pelos desenvolvedores quando há necessidade de integração de protocolos de segurança em seus processos de desenvolvimento;
- Ch7: Desprezo no controle de alterações na segurança (C3 - Conformidade e políticas): realização de uma auditoria adequada em relação às políticas de segurança é algo necessário para o controle de mudanças na organização;
- Ch8: Utilização de métricas não confiáveis (C4 - Estratégia e matrizes): destaca os desafios associados ao uso de métricas de desempenho não confiáveis para avaliar os aspectos de segurança dos processos;
- Ch9: Aplicação de políticas de segurança na elaboração de requisitos (C5 - Requisitos): importância de se realizar a aplicação de políticas de *compliance* na elaboração dos requisitos de cada projeto conduzido;

- Ch10: Falta de conscientização da importância da segurança de software por parte dos desenvolvedores (C6 - Treinamento): refere-se à falta de conscientização e conhecimento dos desenvolvedores quanto à importância da utilização de boas práticas de segurança no processo de desenvolvimento de software;
- Ch11: Integração e entrega de recursos de segurança (C7 - Recurso de Segurança e Design): enfatiza a necessidade de se realizar a integração e fornecer recursos de segurança de forma eficaz nos processos DevOps;
- Ch12: Deficiência de escalabilidade na modelagem de ameaças (C7 - Recurso de Segurança e Design): esse fator aborda os desafios enfrentados na escalabilidade do processo de modelagem de ameaças para atender aos requisitos de projetos;
- Ch13: Revisão de recursos de segurança (C8 - Análise de Arquitetura): destaca a importância de se realizar periodicamente a revisão de recursos de segurança;
- Ch14: Realização de testes periódicos de penetração (C9 - Gerenciamento de Configuração): trata da importância da realização de testes de penetração visando identificar vulnerabilidades na aplicação;
- Ch15: Definição de configurações seguras na fase de implantação (C9 - Gerenciamento de Configuração): aborda os desafios associados à fase de implementação, com a definição de parâmetros e configurações seguras para a implantação do software no DevOps;
- Ch16: Identificação de vulnerabilidades e comunicação com a equipe de desenvolvimento (C9 - Gerenciamento de Configuração): ressalta a importância no fornecimento de *feedback* para a equipe de desenvolvimento, tanto para o tratamento de tais vulnerabilidades, quanto para promover uma maior conscientização futura, reduzindo a chance de uma repetição de tal vulnerabilidade;
- Ch17: Monitoramento de comportamento e geração de diagnósticos das aplicações observadas (c10 - Ambiente de Desenvolvimento): monitoramento de aplicações para detectar eventuais ataques. É ressaltado sobre a necessidade de se gerar relatórios de tais ataques, e
- Ch18: Não utilizar ferramentas precoces (c10 - Ambiente de Desenvolvimento): evitar a utilização de ferramentas de automação imprecisas. Ferramentas precoces são aquelas que fazem orientações erradas aos desenvolvedores em relação à configurações do ambiente.

Os desafios encontrados por Akbar et al. (2022) permitem perceber as dificuldades que o DevSecOps pode trazer para uma equipe de desenvolvimento. Sendo assim, uma

de suas consequências é o atraso no desenvolvimento. Tal conclusão corrobora com os resultados obtidos por Björnholm (2020). Esse atraso é inegável em um primeiro momento, já que com a introdução de novos processos, um atraso na cadeia de desenvolvimento é evidente.

A reflexão que deve ser feita é relacionada à importância na construção de um software seguro. Em outras palavras, refere-se ao quanto a gerência e a equipe estão dispostos a incorrer em uma menor agilidade no desenvolvimento para, no final, obter uma aplicação mais confiável.

Assim, é notável a importância na consolidação e no estabelecimento de políticas claras e de fácil implementação, visando justamente a redução nos atrasos proporcionados pela aplicação de mais uma camada no processo de desenvolvimento.

## 2.6 Considerações Finais do Capítulo

Este capítulo apresentou o referencial teórico do trabalho que possui quatro seções: segurança de software, OWASP, DevOps e DevSecOps. Estes tópicos foram elaborados com base em artigos, livros e materiais anteriormente publicados, englobando conceitos, exemplos e diagramas para apresentação de termos que compreenderam a base fundamental para a produção deste trabalho. Em resumo, deve-ser ter em mente que Segurança de Sofware não deve ser neglicenciada, uma vez que isso pode impactar no sucesso do software como um todo, além de prejuizos que vão além de recursos financeiros. Adicionalmente, há uma compreensão da área sobre quais diretrizes são mais relevantes, e devem ser ponderadas ao longo do ciclo de vida de um software, com destaque para as OWASP Top Ten. Por fim, não se deve adiar a implantação de políticas de segurança. Ao contrário, deve-se trazer essa preocupação mais à esquerda de um *pipeline* de desenvolvimento de software. Sendo assim, cabe pensar em segurança o quanto antes, ainda em tempo de Desenvolvimento e Operações (DevOps), conforme consta em DevSecOps.

# 3 Suporte Tecnológico

## 3.1 Considerações Iniciais do Capítulo

A seguir, são listadas algumas das ferramentas que foram utilizadas pela dupla na elaboração do trabalho. As ferramentas foram subdivididas em: FERRAMENTAS DE APOIO AO DESENVOLVIMENTO, com as ferramentas relacionadas ao desenvolvimento de código; FERRAMENTAS DE APOIO AO *PIPELINE* DEVSECOPS, focando em ferramentas usadas no *pipeline*. Finalmente, são feitas as CONSIDERAÇÕES FINAIS DO CAPÍTULO, disponibilizando uma tabela geral com todas as ferramentas, seus *links* e as suas respectivas versões.

## 3.2 Ferramentas de Apoio ao Desenvolvimento

Seguem as principais ferramentas escolhidas para estabelecimento de um adequado ambiente de desenvolvimento, em termos de gerenciamento de versões e hospedagem, além de plataformas e *frameworks* de desenvolvimento.

### 3.2.1 GitHub

GitHub (GitHub, 2023) é uma plataforma gratuita de hospedagem de projetos, permitindo a criação tanto de projetos públicos, *Open Source*, quanto privados, em repositórios. O GitHub utiliza-se do Git (Git, 2023), que é um sistema de controle de versões distribuido, tornando possível o rastreamento de histórico de alterações de um certo projeto. Foi usado para hospedar e gerenciar a aplicação a ser desenvolvida.

### 3.2.2 Visual Studio Code 1.84

Também conhecido como VSCode, o Visual Studio Code (Visual Studio Code, 2023) é um editor aberto disponível para Windows, Linux e MacOS. Possui suporte ao desenvolvimento de diferentes linguagens de programação, além de prover uma grande quantidade de extensões visando facilitar o processo de desenvolvimento. Foi escolhido pelos desenvolvedores pela familiaridade com a ferramenta, bem como sua popularidade e sua pertinência ao projeto..

### 3.2.3 ReactJS 18.2.0

O ReactJS (React, 2023) é uma biblioteca em JavaScript que proporciona suporte na concepção de interfaces de usuário tanto para aplicações Web quanto nativas, enquanto efetivamente gerencia componentes, estados e propriedades. Com uma comunidade robusta que reúne dois milhões de desenvolvedores mensalmente, o ReactJS solidificou sua posição como uma escolha proeminente no desenvolvimento Frontend. A decisão de optar por essa biblioteca foi motivada por sua ampla adoção e pela familiaridade que os autores possuem em sua utilização.

### 3.2.4 NodeJS 20.9.0

O Node.js (Node, 2023) é uma tecnologia de código aberto que permite executar o JavaScript em servidor. Foi escolhido para o desenvolvimento *Backend* por ser conhecido na comunidade de software e por sua flexibilidade. O NodeJS oferece também um gerenciador de pacotes popularmente conhecido como npm (npm, 2023), ou *Node Package Manager*, que oferece repositórios públicos e privados. Tal suporte permite que desenvolvedores de qualquer parte do mundo disponibilizem diversas ferramentas visando facilitar a implementação de funcionalidades em aplicações desenvolvidas por terceiros. O npm é composto pelo seu *site*, que lista todos os pacotes disponíveis; pela ferramenta CLI (*Command Line Interface*), que permite a instalação dos pacotes, e o *registry*, que atua como um banco de dados centralizado que contém informações sobre diferentes versões de pacotes, suas dependências e seus metadados.

### 3.2.5 ExpressJS 4.18.2

O Express (Express, 2023) é um *framework* do Node.js que fornece recursos para aplicações *Web* e *Mobile*, tornando a criação de APIs robustas um processo rápido e fácil por ser flexível e minimalista. A escolha deste *Framework* para auxiliar o desenvolvimento *Backend* do projeto ocorreu devido à compatibilidade deste com as tecnologias escolhidas, e visando facilitar a criação da API (*Application Programming Interface*). Essa API fez-se necessária, principalmente, na comunicação com os bancos de dados das aplicações desenvolvidas. Exemplos de funcionalidades que possuem interação com as APIs são a criação de conta e a autenticação nas aplicações. Demais funcionalidade são relacionadas ao contexto da própria aplicação. Detalhamentos adicionais sobre as funcionalidades específicas de cada uma constam no Capítulo 6 - Proposta do Trabalho.

### 3.2.6 MaterialUI

O Material UI (MUI, 2023), ou MUI, oferece diversas ferramentas de forma gratuita com o objetivo de facilitar o desenvolvimento de aplicações por meio de variados componentes facilmente customizáveis.

### 3.2.7 Bootstrap

O Bootstrap (BOOTSTRAP JACOB THORNTON, 2024) é um conjunto de ferramentas *Frontend* que facilita o desenvolvimento de aplicações rapidamente e de modo responsivo. O Bootstrap foi empregado nas duas POCs.

## 3.3 Ferramentas de Apoio ao *Pipeline* DevSecOps

Seguem as principais ferramentas para condução das atividades junto ao *pipeline* DevSecOps, cumprindo com as principais demandas desse processo. Este trabalho possui dependência das ferramentas empregadas, e a falta de atualização de uma ferramenta não significa mau funcionamento do *pipeline*. Neste caso, convém testar outras ferramentas *open source* que realizem o mesmo tipo de identificação de vulnerabilidades.

### 3.3.1 Criação de CI/CD

Nesse primeiro momento, há destaque às ferramentas que viabilizam Integração e Entregas Contínuas.

#### 3.3.1.1 GitHub Actions

Para Integração Contínua, o GitHub Actions (GitHub Actions, 2023) é uma funcionalidade disponível na plataforma GitHub que permite a realização de fluxos de trabalhos (*workflows*) de forma automatizada, possuindo uma vasta documentação para sua utilização. Com essa ferramenta, é possível criar e compartilhar *actions* para diversas tarefas. Isso acelera consideravelmente o fluxo de desenvolvimento.

A funcionalidade GitHub Actions oferece fluxos de trabalho que podem ser hospedados em máquinas virtuais do GitHub ou máquinas próprias do usuário que utilizará estas *Actions*. Estes fluxos de trabalho são definidos no diretório .github/workflows em um arquivo de configuração YAML (*YAML Ain't Markup Language*), especificando ações, eventos e ambientes que devem ser executados (ACTIONS, 2023). Os principais conceitos englobados nessa ferramenta são *Events*, *Jobs*, *Steps* e *Actions*.

Os eventos, representados por *Events*, são os principais responsáveis no acionamento do fluxo de trabalho no Gihub Actions. Estes eventos podem ser *push*, *fork*, *pull request* e outros. Há ainda a possibilidade de adição de filtros para definir em qual *branch* o evento deve ocorrer para ativar um *job* específico.

O *Job*, que tem tradução em português como trabalho, compõe-se por um conjunto de tarefas que serão executadas em um computador *executor*. Por padrão, os *Jobs* são executados em paralelo a outros *Jobs*, embora haja a configuração para a execução sequencial. Para cada trabalho, podem ser utilizados conteiners de serviço que são derrubados assim que o *Job* é finalizado (ACTIONS, 2023).

Cada *Job* é constituído pelos *Steps*, onde cada *Step* executa uma tarefa específica e representa uma única unidade de execução. A execução bem sucedida de um *Step* é necessária para o início da execução do próximo. Os *Steps* podem utilizar *actions*, que são unidades de código pré-definidas desenvolvidas pela comunidade ou internamente e adicionadas aos fluxos de trabalho.

A Figura 16 apresenta um exemplo de arquivo .yaml de definição de *Workflow*.

Figura 16 – Exemplo de definição de fluxo de trabalho no *GitHub Actions*

```yaml
name: Octo Organization CI

on:
  push:
    branches: [ $default-branch ]
  pull_request:
    branches: [ $default-branch ]

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v4

      - name: Run a one-line script
        run: echo Hello from Octo Organization
```

Fonte: (ACTIONS, 2023)

Além dos conceitos principais, a ferramenta, para ser utilizada em um contexto de Segurança de Software, deve ser igualmente segura. A segurança pode ser percebida na ferramenta *GitHub Actions* por meio das *SECRETS*, ou segredos, que são utilizados nos arquivos de definição de fluxo de trabalho para armazenar informações sensíveis de forma criptografada e não em texto claro. Algumas práticas recomendadas para manter a segurança nesta ferramenta é registrar todos os segredos que são usados no fluxo; não utilizar dados estruturados para guardar o segredo; verificar se os segredos ainda estão sendo usados, como estão sendo usados e o tempo de validade (ACTIONS, 2023).

Utilizar *CODOWNERS* permite também verificar as alterações realizadas nos arquivos de configuração de fluxo de trabalho. Além disso, para evitar ataques de injeção de *scripts*, pode ser utilizada a ação no lugar do *script* em linha, variáveis de ambiente intermediárias, restrição de permissão para tokens entre outras medidas. Por fim, deve-se estar atento ao utilizar ações de terceiros e fluxos de terceiros entre outras medidas que podem ser adotadas para aumentar a segurança nos *workflows* CI criados. A ferramenta *Github Actions* foi utilizada para a criação do *Pipeline CI* (ACTIONS, 2023).

### 3.3.2 Vercel

O Vercel (VERCEL, 2024) é uma plataforma para criar e implantar aplicativos Web, principalmente *Frontend*, oferecendo estruturas escalonáveis. A escolha do Vercel no *Frontend* deu-se para não utilizar créditos em demasia do Heroku, já que para cada POC, foram feitos os *Deploys* de 4 aplicações (*Frontend* produção, *Frontend* homologação, *Backend* produção e *Backend* homologação). O Vercel possui integração com o CI/CD do Github, e o *Deploy* na ferramenta é simples, além de gratuito .

### 3.3.3 Heroku

Uma ferramenta que fornece suporte à Entrega Contínua é o Heroku (HEROKU, 2023). Essa plataforma permite o *deploy* e o gerenciamento de aplicações em Ruby, Node.js, Java, Python, Clojure, Scala, Go and PHP (HERKOU, 2023). Foi possível utilizar o Heroku integrado ao Git e realizar o *deploy* utilizando o comando "git push heroku main". O Heroku permitiu definir etapas personalizadas para as fases do *pipeline*.

As típicas etapas adicionadas são o *Build*, *Testes* e *Deploy*. Durante estas etapas, o Heroku pode provisionar automaticamente recursos adicionais necessários, como bancos de dados, cache e filas de mensagens. O Heroku pode fornecer também roteamento, e a aplicação é, então, empacotada em *dynos* (os conteiners executáveis do Heroku), que são compilados e iniciados para gerar a página do *Deploy* fornecido por ele.

O Heroku disponibiliza créditos para estudantes, e estes créditos foram utilizados para gerar o *Deploy*.

#### 3.3.3.1 Docker

O Docker (Docker, 2021) foi introduzido em 2013 na indústria, e permite conteinerização da aplicação, isolando-a do ambiente e facilitando a execução do código fonte em máquinas com configurações distintas e pacotes com versões diferentes do que é exigido pela aplicação. A utilização de contêineres oferece maior isolamento do ambiente de

execução da aplicação, assegurando que não haja interferência em outras máquinas. Além disso, proporciona portabilidade, permitindo transferência fácil para diferentes ambientes. A escalabilidade é simplificada, facilitando atualizações e mantendo padrões, o que contribui para a automação e integração contínua.

A execução do código em um contêiner requer a criação de um arquivo chamado Dockerfile, que detalha o conteúdo do contêiner e fornece instruções para construir uma imagem, necessária para executar o contêniner. Este arquivo inclui dependências, comandos de configuração e a própria aplicação. Utiliza-se também o arquivo compose.yml, definindo como executar e gerar uma instância da imagem. É possível fazer o *build* das imagens Docker a partir de comandos fornecidos nas definições de Workflow do *GitHub Actions* (DOCKER, 2023).

Para orquestrar múltiplos contêineres em aplicações mais complexas, há a ferramenta Docker Compose que, por meio do docker-compose.yml, define imagens Docker, configurações de rede, variáveis de ambiente, volumes e outros parâmetros para instanciar o ambiente desejado.

Para assegurar a segurança dos contêineres, há um conjunto de boas práticas a serem seguidas. Estas incluem a escolha de imagens de uma base de dados confiável; a preferência por imagens base pequenas para minimizar vulnerabilidades de dependências, e a utilização de imagens distintas em ambientes de desenvolvimento e produção, empregando somente o essencial. Além disso, reconstruir imagens regularmente usando o comando *build* com a opção –no-cache é recomendado para evitar acessos ao cache. Por fim, o uso do Docker Scout CLI possibilita a exploração de vulnerabilidades de imagens por meio de terminal (DOCKER, 2023).

O Docker foi escolhido devido às características de portabilidade e facilidade de integração com o software e as tecnologias selecionadas.

### 3.3.4 Ferramentas SAST

As subseções a seguir focam nas ferramentas de testes estáticos. A característica principal de um teste estático é não precisar executar o código para poder testá-lo.

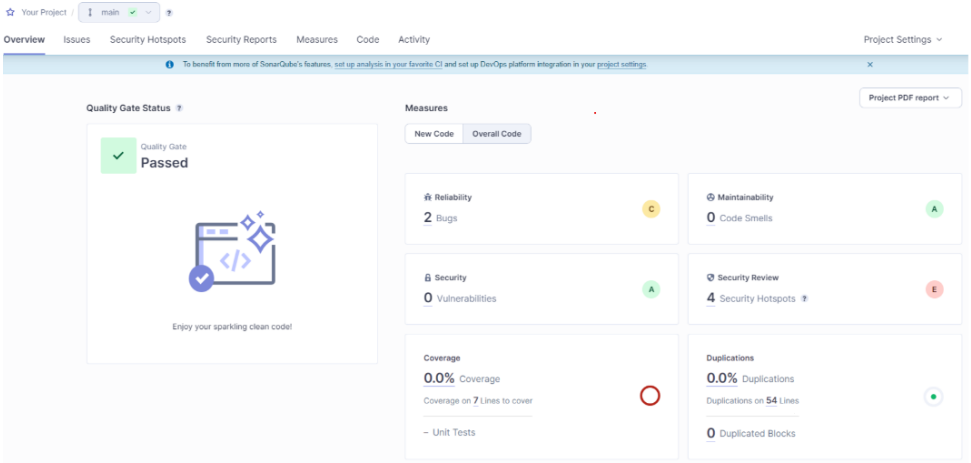
#### 3.3.4.1 SonarQube

O SonarQube (SonarQube, 2023) faz uma análise estática e de qualidade do código-fonte do projeto. Esta ferramenta foi acoplada ao *pipeline* DevSecOps para análise de qualidade antes da implementação do código em ambiente de produção.

O Sonarqube possui um *Sonar way quality profile* que se refere a uma série de métricas e regras a serem seguidas por um projeto, podendo ser configurável para cada um dependendo das necessidades específicas de cada um. Com essas regras configuradas, o Sonarqube analisa a qualidade do código por meio do *Quality Gate*, que é um indicador de qualidade que pode ser vermelho, indicando que há problemas a serem resolvidos; ou verde, indicando que o código está padronizado e pode ser integrado ao código principal. Os problemas identificados no *SonarQube* estão relacionados aos atributos agrupados nas categorias consistente, intencional, adaptável e responsável (SONARQUBE, 2023).

A Figura 17 apresenta a interface do Sonarqube e algumas análises feitas pela ferramenta.

Figura 17 – *Quality Gate* e análise do código fonte no SonarQube




Fonte: (SONARQUBE, 2023)

As métricas analisadas pelo Sonarqube são (SONARQUBE, 2023):

- Complexidade: refere-se à complexidade ciclomática do código. Essa métrica avalia quantos caminhos diferentes o código pode seguir a partir de instruções condicionais;
- Duplicações: número de blocos de linhas duplicadas, é composto pelos seguintes elementos: arquivos duplicados, linhas duplicadas e porcentagem de linhas duplicadas. Esse indicador quantifica a presença de duplicações no código-fonte, identificando blocos de código semelhantes ou idênticos ao longo do projeto;
- Issues: refere-se ao controle de questões relacionadas ao código-fonte, incluindo criação, início, confirmação e reabertura. Esse controle permite compreender o fluxo de trabalho;
- Manutenibilidade: esta métrica é composta por *code smells*, novos *code smalls*, *rating*, e débitos técnicos relacionados à correção de *code smells*;

- Confiabilidade: é composta pela quantidade total de *bugs*, novos *bugs* e por um *rating* que indica a confiabilidade do código, fundamentado na quantidade de *bugs* identificados;
- Segurança: essa métrica abrange o número total de vulnerabilidades; vulnerabilidades em novos códigos adicionados; o *rating* de segurança (A = 0 Vulnerabilidades, B = pelo menos 1 Vulnerabilidade Menor, C = pelo menos 1 Vulnerabilidade Major, D = pelo menos 1 Vulnerabilidade Crítica, E = pelo menos 1 Vulnerabilidade Bloqueadora); o esforço para corrigir vulnerabilidades; o esforço para corrigir vulnerabilidades em novos códigos; os *security hotspots* encontrados no código, gerando um *rating* relacionado à meta de porcentagem de *security hotspots* solucionados, e o mesmo *rating* para novos códigos, *security hotspots* revisados e novos *security hotspots* revisados. Os *Security Hotspots* são trechos sensíveis à segurança, porém, não necessariamente influenciam na segurança de toda a aplicação. As vulnerabilidades, por outro lado, influenciam na segurança de toda a aplicação e devem ser corrigidas imediatamente após identificadas;
- Tamanho: métrica referente ao tamanho do código fonte, considerando número de diretórios, arquivos, funções, classes, linhas, linhas comentadas e, outros, e
- Testes: métrica que reflete a quantidade de código com testes aplicados, chamado *coverage*. Quanto maior o *coverage*, mais funções estão sendo testadas.

Com base nestas métricas e na meta definida para cada projeto, é calculado o *Quality Gate*. Além da segurança no código fonte, a ferramenta deve ser considerada segura para utilização. Esta segurança é vista uma vez que o *SonarQube* possui mecanismos de autenticação e autorização integrados, e criptografia de configurações e senhas. O Sonarqube foi selecionado por ser automático, autogerenciável e compatível com o GitHub Actions. Além disso, confere análise detalhada de qualidade; facilidade na configuração, e documentação abrangente disponível (SONARQUBE, 2023).

#### 3.3.4.2 CodeQL

O CodeQL (CODEQL, 2024) faz a análise semântica do código e pode ser habilitado no Github. É uma ferramenta gratuita para pesquisa e código aberto e é configurada no Github.

### 3.3.5 Ferramentas SCA

Segue a ferramenta que conferiu suporte ao processo de identificação de vulnerabilidades.

#### 3.3.5.1 Owasp-Dependency-Check 9.0.0

O OWASP *Dependency-Check* (OWASP Dependency Check, 2023) é uma ferramenta *open source* desenvolvida pela OWASP. Surgiu para evitar a Top 10 Vulnerabilidade OWASP A06:2021 – *Vulnerable and Outdated Components*, e identificar vulnerabilidades conhecidas de componentes. O objetivo desta ferramenta é detectar dependências vulneráveis utilizadas na aplicação alvo. Caso encontre, é gerado um relatório com as devidas referências para a vulnerabilidade (OWASP Dependency Check, 2023).

Esta ferramenta utiliza analisadores para coletar informações presentes em arquivos de configuração como o "package.json" e gerar, a partir destas informações, as chamadas evidências. As evidências são fornecedor, nome do produto e versão da dependência. Além disso, esta ferramenta tem integração com o Github e SonarQube. A Figura 18 apresenta um exemplo de utilização desta ferramenta em conjunto com a ferramenta Docker (OWASP Dependency Check, 2023).

Figura 18 – Exemplo de OWASP *Dependency-Check*

```
#!/bin/sh

DC_VERSION="latest"
DC_DIRECTORY=$HOME/OWASP-Dependency-Check
DC_PROJECT="dependency-check scan: $(pwd)"
DATA_DIRECTORY="$DC_DIRECTORY/data"
CACHE_DIRECTORY="$DC_DIRECTORY/data/cache"

if [ ! -d "$DATA_DIRECTORY" ]; then
    echo "Initially creating persistent directory: $DATA_DIRECTORY"
    mkdir -p "$DATA_DIRECTORY"
fi
if [ ! -d "$CACHE_DIRECTORY" ]; then
    echo "Initially creating persistent directory: $CACHE_DIRECTORY"
    mkdir -p "$CACHE_DIRECTORY"
fi

# Make sure we are using the latest version
docker pull owasp/dependency-check:$DC_VERSION

docker run --rm \
    -e user=$USER \
    -u $(id -u ${USER}):$(id -g ${USER}) \
    --volume $(pwd):/src:z \
    --volume "$DATA_DIRECTORY":/usr/share/dependency-check/data:z \
    --volume $(pwd)/odc-reports:/report:z \
    owasp/dependency-check:$DC_VERSION \
    --scan /src \
    --format "ALL" \
    --project "$DC_PROJECT" \
    --out /report
    # Use suppression like this: (where /src == $pwd)
    # --suppression "/src/security/dependency-check-suppression.xml"
```

Fonte: (OWASP Dependency Check, 2023)

Essa ferramenta foi selecionada, pois está em coerência com as OWASP Top Ten diretrizes de cibersegurança.

#### 3.3.5.2 Snyk *Open Source*

A análise das dependências vulneráveis de código também foi realizada pela ferramenta Snyk, em sua versão *Open Source* (SNYK, 2024). A ferramenta analisa bibliotecas de código aberto em busca de vulnerabilidades. O Snyk suporta JavaScript, Python, .NET, PHP e Go.

#### 3.3.5.3 Dependabot

O Dependabot (DEPENDABOT, 2024) é um recurso do próprio Github, e a funcionalidade habilitada no *pipeline* foi a funcionalidade de alertas do Dependabot, que informam as dependências vulneráveis da aplicação.

#### 3.3.5.4 *GitGuardian*

O GitGuardian (GITGUARDIAN, 2024) detecta senhas, segredos, chaves de api e outros dados confidênciais. Seu emprego no *pipeline* foi na indicação destes dados.

### 3.3.6 Ferramentas DAST

As subseções a seguir focam nas ferramentas de testes dinâmicos.

#### 3.3.6.1 ZAP 2.14.0

ZAP (ZAP, 2023) é uma ferramenta para testes de segurança de código aberto para aplicações Web. Foi desenvolvida pela OWASP. Esta ferramenta fornece varreduras automatizadas, interface gráfica e relatórios. A ferramenta ZAP fica entre o navegador e a aplicação Web (cliente e servidor) para interceptar e inserir conteúdo nas mensagens trocadas entre estas camadas, para só então encaminhar para o destino. O ZAP não está vinculado a sistemas operacionais, podendo atuar como *stand- alone* ou *daemon* (ZAP, 2023).

A ZAP foi escolhida para o projeto para a aplicação de testes dinâmicos de segurança no *pipeline* DevSecOps.

#### 3.3.6.2 SQLMap 1.7

O SQLMAP (SQLMAP, 2023) é uma ferramenta de testes de penetração de código aberto, utilizada para identificar falhas de SQLInjection e controle de servidores de banco de dados em uma aplicação, apresentando relatórios sobre as vulnerabilidades encontradas. Oferece suporte a diferentes bancos, entre eles o MySQL, PostgreSQL, Microsoft SQL Server, Oracle, SQLite, entre outros. Consegue encontrar seis técnicas de injeção SQL, sendo elas, *boolean-based blind*, *time-based blind*, *error-based*, *UNION query-based*, *stacked queries* and *out-of-band* (SQLMAP, 2023). A Figura 19 apresenta um exemplo de utilização da ferramenta.

Figura 19 – Exemplo de código de ataque SQLMAP

```
stamparm@beast:~/Dropbox/Work/sqlmap$ python sqlmap.py -u "http://debiandev/sqlmap/mysql/g
et_int.php?id=1" --batch --passwords
```

Fonte: (SQLMAP, 2023)

Após chamar a ferramenta e passar os parâmetros necessários, é possível visualizar os resultados. A Figura 20 apresenta os resultados do teste de penetração.

Figura 20 – Exemplo de resultado do ataque com SQLMAP

```
[17:22:17] [INFO] the back-end DBMS is MySQL
web application technology: PHP 5.2.6, Apache 2.2.9
back-end DBMS: MySQL 5.0
[17:22:17] [INFO] fetching database users password hashes
do you want to store hashes to a temporary file for eventual further processing with other
 tools [y/N] N
do you want to perform a dictionary-based attack against retrieved password hashes? [Y/n/q
] Y
[17:22:17] [INFO] using hash method 'mysql_passwd'
what dictionary do you want to use?
[1] default dictionary file '/home/stamparm/Dropbox/Work/sqlmap/txt/wordlist.zip' (press E
nter)
[2] custom dictionary file
[3] file with list of dictionary files
> 1
[17:22:17] [INFO] using default dictionary
do you want to use common password suffixes? (slow!) [y/N] N
[17:22:17] [INFO] starting dictionary-based cracking (mysql_passwd)
[17:22:17] [INFO] starting 8 processes
[17:22:20] [INFO] cracked password 'testpass' for user 'root'
database management system users password hashes:
[*] debian-sys-maint [1]:
    password hash: *6B2C58EABD91C1776DA223B088B601604F898847
[*] root [1]:
    password hash: *00E247AC5F9AF26AE0194B41E1E769DEE1429A29
    clear-text password: testpass

[17:22:21] [INFO] fetched data logged to text files under '/home/stamparm/.sqlmap/output/d
ebiandev'
stamparm@beast:~/Dropbox/Work/sqlmap$
```

Fonte: (SQLMAP, 2023)

O resultados dos testes apresentam as senhas que não deveriam ser acessadas pelo usuário. Esta ferramenta foi escolhida para adicionar testes de penetração no *pipeline* DevOps do software.

#### 3.3.6.3 Nikto 2.1.5

O Nikto (sullo, 2023) é um scanner de vulnerabilidades de código livre, desenvolvido para utilização em CLI (*Command Line Interface*). Essa ferramenta verifica as versões desatualizadas de servidores e os arquivos de configuração. O Nikto testa um servidor Web no menor tempo posível para encontrar possíveis problemas como configurações incorretas; arquivos e programas padrão ou inseguros, e análises de resposta do servidor scaneando uma ou mais portas de servidores. Gera arquivos de saída contendo os relatórios customizáveis que podem ter formato csv, html, xml, nbe, json ou sql (sullo, 2023).

É comumente utilizado em conjunto com outras ferramentas. Seu objetivo é analisar servidores web em busca de vulnerabilidades em servidores ao nível de configuração.

## 3.4 Considerações Finais do Capítulo

Este capítulo descreveu as principais tecnologias utilizadas neste trabalho. A Tabela 1 possui o nome de cada ferramenta, sua finalidade, o *link* de acesso e a versão utilizada.

Tabela 1 – Ferramentas de apoio ao trabalho

| Ferramenta | Finalidade | Link | Versão |
|---|---|---|---|
| GitHub | Armazenamento e controle de versão | https://github.com/ | - |
| Visual Studio Code | Escrita do código-fonte | https://code.visualstudio.com/ | 1.84 |
| ReactJS | Desenvolvimento do *front-end* | https://react.dev/ | 18.2.0 |
| NodeJS | Desenvolvimento do *back-end* | https://nodejs.org/en | 20.9.0 |
| ExpressJS | Framework para construção do servidor web | https://expressjs.com/pt-br/ | 4.18.2 |
| MaterialUI | Biblioteca de componentes para ReactJS | https://mui.com/material-ui/ | 5.14.18 |
| Bootstrap | Ferramenta gratuita para desenvolvimento HTML, CSS e JS | https://getbootstrap.com.br/ | - |
| GitHub Actions | Execução das *pipelines* | https://github.com/features/actions/ | - |
| Vercel | *Deploy* da aplicação | https://vercel.com/ | - |
| Heroku | *Deploy* da aplicação | https://www.heroku.com/ | - |
| Docker | Conteinerização da aplicação | https://www.docker.com/ | - |
| Checkmarx | Ferramenta de análise estática | https://checkmarx.com/ | - |
| SonarQube | Ferramenta de análise estática | https://www.sonarsource.com/products/sonarqube/ | - |
| CodeQL | Ferramenta de análise estática | https://codeql.github.com/ | - |
| Owasp-Dependency-Check | Ferramenta para análise da composição do código | https://owasp.org/www-project-dependency-check/ | 9.0.0 |
| Snyk *Open Source* | Ferramenta para análise da composição do código | https://snyk.io/ | - |
| Dependabot | Ferramenta para análise da composição do código | https://github.com/dependabot | - |
| GitGuardian | Ferramenta para detecção de dados sensíveis | https://www.gitguardian.com/ | - |
| ZAP | Ferramenta de análise dinâmica | https://www.zaproxy.org/ | 2.14.0 |
| SQLMap | Ferramenta de análise dinâmica | https://github.com/sqlmapproject/sqlmap | 1.7 |
| Nikto | Ferramenta de análise dinâmica | https://github.com/sullo/nikto | 2.1.5 |

Fonte: Autores

# 4 Metodologia

## 4.1 Considerações Iniciais do Capítulo

Este capítulo apresenta o detalhamento metodológico utilizado no desenvolvimento teórico e prático desta monografia. Primeiramente, é feita a CLASSIFICAÇÃO DE PESQUISA, na qual a pesquisa é classificada quanto à abordagem, quanto à natureza, quanto aos objetivos e quanto aos procedimentos. Após isso, é detalhado o MÉTODO INVESTIGATIVO utilizado para o embasamento trabalho, e para a condução da pesquisa bibliográfica. O principal método utilizado neste trabalho é apresentado no MÉTODO INVESTIGATIVO, juntamente com o MÉTODO DE DESENVOLVIMENTO e o MÉTODO DE ANÁLISE DE RESULTADOS utilizados nas Provas de Conceito. Em seguida, são apresentados os FLUXOS DE ATIVIDADES e cronogramas do projeto. Por último, são apresentadas as CONSIDERAÇÕES FINAIS DO CAPÍTULO, onde é feito um resumo do que foi abordado no capítulo.

## 4.2 Classificação de Pesquisa

Uma pesquisa é viabilizada por meio de aproximações sucessivas da realidade, e a metodologia utilizada refere-se ao corpo de regras e diligências estabelecidas para executá-la cientificamente (GERHARDT; SILVEIRA, 2009).

Nesse sentido, como apresentado por Gerdhardt e Silveira (2009), foi realizada a classificação da pesquisa deste trabalho quanto aos aspectos de abordagem, natureza, objetivos e procedimentos, conforme sintetizado no Quadro 1.

Quadro 1 – Classificação da pesquisa

| | |
|---|---|
| Abordagem | Qualitativa e Quantitativa |
| Natureza | Aplicada |
| Objetivos | Exploratória |
| Procedimentos | Provas de Conceito |

Fonte: Autoria própria.

### 4.2.1 Abordagem da Pesquisa

Quanto à abordagem, esta pesquisa pode ser classificada como pesquisa **qualitativa** e **quantitativa**. A abordagem qualitativa resulta da análise de alguns dados

não-métricos que se valem de diferentes formulações, centrando-se na descrição, compreensão e explicação de um determinado fenômeno (GERHARDT; SILVEIRA, 2009). Já a abordagem qualitativa refere-se à coleta e à análise de dados numéricos (GERHARDT; SILVEIRA, 2009).

Neste caso, conforme especificado no Método de Análise de Resultados, a análise qualitativa encontra-se na qualidade e na segurança do código-fonte, além da própria avaliação do *pipeline* e do detalhamento das ferramentas utilizadas. Esta pesquisa é também quantitativa, uma vez que é feita a interpretação de dados numéricos para determinação da eficácia do *pipeline* DevSecOps.

Em termos qualitativos, portanto, foi reportado se o código, por exemplo: atende/satisfaz às diretrizes OWASP Top Ten, ou ainda se possui vulnerablidades. Por serem critérios qualitativos, foi ponderado o quanto atende, ou o quanto possui utilizando a escala Likert. Proposta por Rensis Likert (1932), trata-se de uma escala de qualificação, a qual faz uso de "inquéritos" para questionar sobre o alvo investigado, revelando o seu nível de acordo ou desacordo com base em um determinado aspecto. Foram usados 5 níveis de ponto, mantendo um neutro, no intuito de permitir registros sem posicionamento claro sobre o que se encontra em investigação. Sendo assim, no caso de atender/satisfazer ou não as diretrizes OWASP, os registros não ficaram limitados a sim ou não. Na verdade, ocorreu algo como: atende/satisfaz, atende/satisfaz razoavelmente, sem opinião formada, atende/satisfaz pouco, não atende/satisfaz. Isso possibilitou aferir aspectos subjetivos, não quantificáveis (LIKERT, 1932).

Em termos quantitativos, foram utilizadas métricas mensuáveis, com base em ferramentas de testes estáticos, dinâmicos, inspeção por revisão por pares e testes de tistema. Algumas dessas métricas são descritas mais adiante, nesse mesmo capítulo, na seção Método de Análise de Resultados.

### 4.2.2 Natureza da Pesquisa

A natureza desta pesquisa é **aplicada** uma vez que, segundo Gerdhardt e Silveira (2009), soluciona problemas específicos e produz conhecimento para aplicação prática.

### 4.2.3 Objetivos da Pesquisa

Esta pesquisa pode ser classificada como **exploratória**, visto que proporcionou maior familiaridade com o problema para torná-lo mais explícito ou construir hipóteses em cima dele (GIL, 2017).

Este caráter exploratório residiu na busca por compreender, experimentar e testar novas abordagens para integrar a segurança no ciclo de vida do desenvolvimento de software, evoluindo o *pipeline*, formulando e validando a eficácia das hipóteses elaboradas.

### 4.2.4 Procedimentos da Pesquisa

Quanto aos procedimentos, este trabalho utilizou o Método de **Provas de Conceito**, com o objetivo de investigar um tema em específico criando provas de conceito que comprovam o funcionamento do *pipeline* proposto.

Com base em dois domínios (domínio financeiro e domínio bancário), foram identificados cenários de uso para implementação de aplicações. Cada cenário de uso foi tratado orientando-se por POCs. Isso permitiu desenvolver cada POC utilizando uma abordagem DevSecOps. Ao final de cada POC, foi disponibilizada a análise qualitativa/quantitativa, conferindo insumos para responder as Questões de Pesquisa e Questão de Desenvolvimento desse trabalho. Essa estratégia auxiliou na modularização do problema, possibilitando lidar com cada etapa do *pipeline* DevOps, evidenciando preocupações de segurança, com a identificação de vulnerabilidades. Na sequência, cada vulnerabilidade foi tratada usando a abordagem DevSecOps e as OWASP Top Ten. Com isso, foi possível observar a aplicabilidade e os resultados da inserção do processo centrado em Segurança de Software desde às etapas que antecedem a implantação do software.

Como procedimento complementar, a **Pesquisa-Ação** foi utilizada em um segundo momento para análise de dados e discussão de resultados. Esse método supõe uma forma de ação de caráter técnico, diante da análise de insumos coletados com as provas de conceito. A Pesquisa-Ação apresenta etapas como a fase exploratória; formulação do problema; construção de hipóteses; realização do seminário; seleção da amostra; coleta de dados; análise e interpretação dos dados; elaboração do plano de ação e divulgação dos resultados (GIL, 2017). Demais detalhes, inerentes a esse método, serão revelados na seção 4.6, ainda nesse capítulo.

## 4.3 Método Investigativo

A pesquisa foi conduzida primariamente utilizando o Google Scholar como a principal base de dados acadêmica, com o intuito de identificar uma literatura abrangente composta por materiais divulgados por empresas renomadas, bem como livros e artigos de autores destacados na área de interesse deste trabalho. No entanto, também foram exploradas as bases ACM e IEEE para complementar a busca.

Para a condução da pesquisa, foram delineadas as seguintes *strings* de busca: “Devops”, “Owasp”, “DevSecOps” e “Software Security”. Paralelamente, realizou-se a análise da bibliografia de artigos e livros de autores reconhecidos no campo, com o objetivo de enriquecer a fundamentação teórica.

Adicionalmente, implementou-se o método de *Snowballing* sobre os artigos e livros identificados, buscando expandir a pesquisa. Com isso, foi realizado um refinamento dos artigos encontrados.

Os critérios de seleção utilizados neste refinamento foram:

- Estar escrito em português ou inglês;
- Apresentar resumo acerca dos temas de interesse deste trabalho;
- Tratar de definições de DevOps e DevSecOps;
- Abordar Segurança de Software, e
- Explicar e abordar a OWASP e as Top 10 vulnerabilidades.

Deste modo, os principais materiais selecionados foram:

- *Cryptography and network security: principles and practice.* (STALLINGS, 2017)
- *OWASP Top Ten* (OWASP Foundation, 2003)
- *The DevOps Handbook: How to Create World-Class Agility, Reliability, & Security in Technology Organizations.*, (KIM et al., 2016)
- *The Phoenix Project: A Novel About IT, DevOps, and Helping Your Business Win* (KIM; BEHR; SPAFFORD, 2018),
- *A Software Architect's Perspective* (BASS; WEBER; ZHU, 2015)
- *Securing DevOps: security in the Cloud* (VEHENT, 2018)

O Zotero (2023) foi utilizado para o fichamento dos trabalhos de referência, com especificação dos principais pontos de fundamentação teórica acordados no Capítulo 2 - Referencial Teórico.

## 4.4 Método Geral Orientado a Provas de Conceito

Após a investigação do tema deste trabalho, optou-se por adotar o método orientado a Provas de Conceito (PoCs) para comprovação das hipóteses construídas e verificação da viabilidade do *pipeline* através de *feedbacks* rápidos. As Provas de Conceito (PoCs) podem ser ferramentas essenciais no contexto de desenvolvimento, validando um conceito e demonstrando a capacidade do software em atender às demandas (PRASANNA et al., 2021). A abordagem sistemática das PoCs fornece compreensões sobre a aplicabilidade e a funcionalidade do software. Neste cenário, visando garantir maior esclarecimento acerca de cada PoC, foi elaborado um roteiro especificando os objetivos principais da PoC, vul-

nerabilidades abordadas, definições de ambiente de execução, configurações e implantação da solução.

Antes de iniciar a execução do *pipeline* nas Provas de Conceito, o *pipeline* foi delineado com os insumos necessários para a compreensão. Posteriormente, procedeu-se à aplicação do *pipeline* nas Provas de Conceito. A estrutura proposta para o roteiro da Prova de Conceito (PoC) apresenta uma abordagem organizada em seções para a condução do estudo. A primeira seção é a Definição da PoC. A segunda seção descreve a Solução. Por fim, a terceira seção refere-se à Análise de Resultados.

A primeira seção, Definição da PoC, é composta pelos requisitos necessários na implementação da PoC; o sistema objeto de estudo, com a descrição da POC desenvolvida, código fonte e artefatos necessários para compreensão; e as vulnerabilidades OWASP Top 10 estudadas.

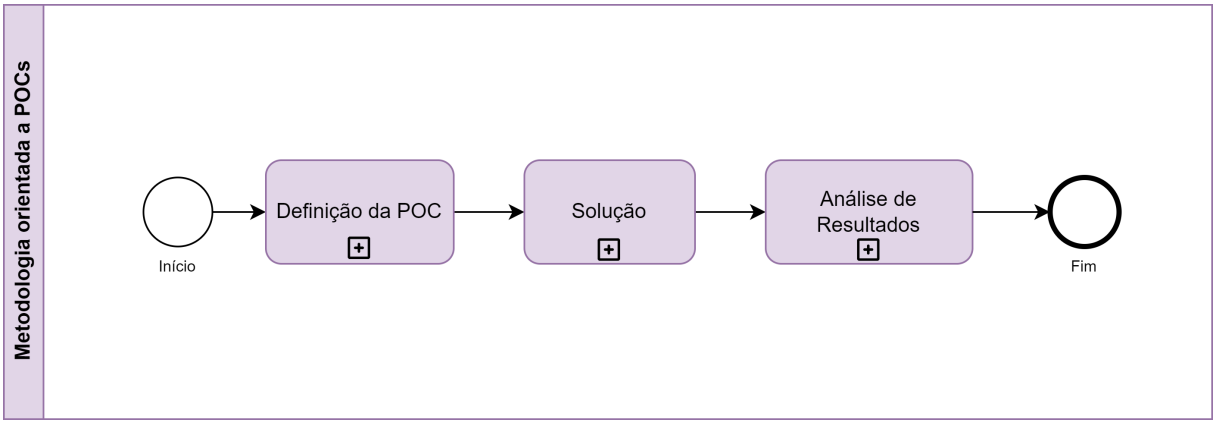
A segunda seção, Solução, possui duas etapas, a aplicação do *pipeline* com descrição da solução implementada para mitigar a vulnerabilidade especificada e alcançar os objetivos propostos e a resolução de vulnerabilidades, que dispõe as etapas de resolução das vulnerabilidades encontradas. O Método de Desenvolvimento foi empregado nesta fase da POC.

Nesta fase de solução algumas vulnerabilidades não podem ser detectadas por ferramentas automatizadas, exigindo uma abordagem semiautomatizada com intervenção de programadores. O método empregado foi o de inspeção com revisão por pares, utilizando a ferramenta GitHub, que permite acesso simultâneo ao código pelos desenvolvedores. Dependendo da profundidade e o tempo despendido para a análise, essa abordagem pode levar os programadores a considerar falsos positivos e realizar refatorações desnecessárias (ALBUQUERQUE et al., 2014). Para otimizar o tempo e evitar refatorações sem ganho real de qualidade, a revisão foi conduzida de forma ampla.

A última seção, análise de resultados, apresenta a avaliação quantitativa e qualitativa da solução descrita na seção anterior, considerando métricas definidas no Método de Análise de Resultados.

A Figura 21 apresenta o fluxo, em modelagem BPMN, das etapas envolvidas na Prova de Conceito.

Figura 21 – Método geral orientado a provas de conceito




Fonte: Autoria própria.

Um método complementar utilizado no desenvolvimento das PoCs foi a Pesquisa-Ação. Este método supõe uma ação de caráter técnico, e conta com o envolvimento ativo do pesquisador (GIL, 2017). As fases exploratória, de formulação do problema, construção de hipóteses e seleção da amostra foram realizadas para compreensão do escopo e elaboração do embasamento teórico do trabalho. As fases de coleta, análise de dados e ação foram utilizadas na melhoria do *pipeline* em cada PoC.

## 4.5 Método de Desenvolvimento

O método de desenvolvimento utilizado no desenvolvimento do projeto foi hibrído, combinando elementos de diferentes metodologias consolidadas na Engenharia de Software. No caso, foi escolhida uma orientação com base em métodos ágeis, sendo: Scrum, XP e Kanban.

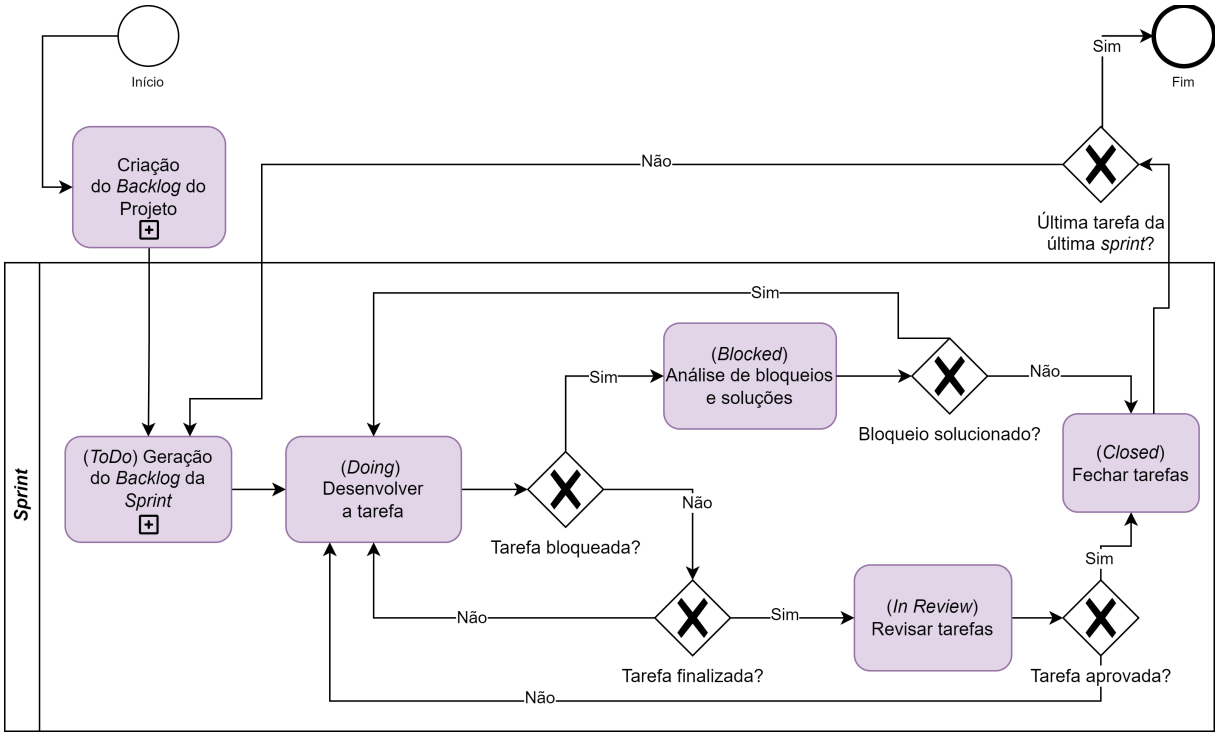
O Scrum define o processo de desenvolvimento de sistemas como um conjunto solto de atividades que combina técnicas viáveis com o melhor que uma equipe de desenvolvimento pode conceber (SUTHERLAND, 1995). Esta metodologia ajuda equipes, pessoas e organizações a gerar valor para problemas com soluções adaptativas. No desenvolvimento das PoCs, foram aplicados elementos como *Sprints* (que se referem a períodos de tempo cíclicos e definidos); *Product Backlog* (abordando alguns conceitos como épico, tarefas e critérios de aceitação), e *Sprint Backlog* (SCHWABER; SUTHERLAND, 2020) juntamente com a condução de cerimônias de *Sprint Planning* e *Sprint Review* para identificar melhorias e planejar os próximos ciclos de desenvolvimento.

O método Kanban incorporado ao projeto permitiu maior visibilidade do fluxo de desenvolvimento de software, facilitando a identificação de bloqueios e prioridades (ANDERSON, 2010). Alinhado a esta abordagem, foi utilizado o método XP, definido por

valores, princípios e práticas de desenvolvimento. Práticas específicas do XP no processo de desenvolvimento, tais como histórias de usuário, *releases* e práticas de programação, incluindo a propriedade coletiva do código, *build* automatizado, integração contínua e programação em pares foram parte do desenvolvimento das PoCs (BECK, 1999).

A junção destes métodos permitiu a construção de um processo com etapas definidas durante o desenvolvimento. A Figura 22 apresenta o fluxo de desenvolvimento deste trabalho.

Figura 22 – Método de desenvolvimento




Fonte: Autoria própria.

O método de desenvolvimento seguiu, em cada *sprint*, as seguintes etapas até a finalização das PoCs:

1. Foi criado um *backlog*, contendo a lista de funcionalidades a serem desenvolvidas na *sprint*, em formato de história de usuário. Estas tarefas iniciaram na coluna *ToDo* do quadro Kanban.

2. Durante a realização das tarefas, elas foram transferidas para a coluna *Doing*, e foram realizadas tanto em *pair programming* quanto individualmente, considerando a complexidade da tarefa.

3. As tarefas bloqueadas, foram movidas para a coluna *Blocked*, facilitando a visualização dos bloqueios na *sprint*.

4. As tarefas foram revisadas, verificadas e validadas para garantir a coerência do que foi desenvolvido, e por isso, ficaram na coluna *In Review* até serem revisadas.

5. Após a revisão, as tarefas foram finalizadas e finalmente alteradas para a coluna *Closed* do quadro Kanban. As tarefas cujos bloqueios não puderam ser solucionados ou foram identificados outros gargalos também foram transferidas para esta coluna.

## 4.6 Método de Análise de Resultados

Nas subseções a seguir, serão considerados dois detalhamentos de relevância para a plena realização da análise de resultados, sendo: métricas, que conferem insumos mais concretos dos resultados obtidos, e protocolo adaptado de pesquisa-ação.

### 4.6.1 Métricas para Análise do *Pipeline* DevSecOps

Considerando a solução desenvolvida e a adaptabilidade intrínseca das Provas de Conceito (POCs) a diversos contextos, a análise de dados neste trabalho foi passível de condução mediante distintas métricas, as quais foram selecionadas de acordo com as especificidades de cada prova de conceito e as necessidades identificadas. Adicionalmente, em circunstâncias pertinentes, foi implementado o ciclo de pesquisa-ação, complementar à análise métrica, para uma abordagem mais abrangente e aplicada, conforme demandado pelo escopo da pesquisa.

No que tange às métricas, a taxa de identificação de vulnerabilidades implantadas reflete a quantidade de vulnerabilidades OWASP TOP 10 implantadas pelos autores que foram corretamente identificadas pela *pipeline*.

$$\text{taxa de vulnerabilidades verificadas} = \frac{\text{quantidade de vulnerabilidades encontradas pelo } \textit{pipeline}}{\text{quantidade de vulnerabilidades implantadas pelos autores no desenvolvimento da POC}} \quad (1)$$

A taxa de vulnerabilidades verificadas reflete a eficácia do *pipeline*, contida no intervalo de 0 a 1, reflete a capacidade do *pipeline* de segurança em identificar vulnerabilidades no contexto do ambiente da POC.

Uma proximidade à extremidade inferior (próxima de 0) sugere uma detecção insuficiente, indicando a identificação de poucas ou nenhuma vulnerabilidade implantada, o que caracteriza um desempenho inadequado do *pipeline* para a aplicação em questão. Em situações em que o valor resultante se revela próximo de 0, os autores têm a prerrogativa de

conduzir uma análise aprofundada do *pipeline* e buscar aprimoramentos visando alcançar resultados mais satisfatórios.

Contrariamente, uma métrica que se aproxima de 1 indica que a grande maioria, ou todas as vulnerabilidades implantadas, foi/foram corretamente identificada(s) pelo *pipeline*. Neste cenário, o *pipeline* é considerado eficaz para o propósito estipulado.

A interpretação formal desta métrica sugere que, quanto maior o valor resultante, mais eficaz é o *pipeline* na detecção das vulnerabilidades implementadas.

Empregar exclusivamente a taxa de vulnerabilidades verificadas não é suficiente para determinar a eficácia do *pipeline*. Para garantir dados mais representativos da realidade, a métrica 2 define quantas vulnerabilidades foram encontradas pelo *pipeline* em cada repositório das Provas de Conceito.

vulnerabilidades encontradas = quantidade de vulnerabilidades encontradas pelo *pipeline* (2)

Com a métrica de vulnerabilidades encontradas, sabe-se quantas vulnerabilidades o *pipeline* encontrou em cada Prova de Conceito.

Adicionalmente à métrica previamente mencionada, foram incorporadas outras métricas qualitativas e quantitativas. Prates (2019) delineou nove métricas para médir a eficácia do DevSecOps. Entretanto, algumas foram adaptadas e outras foram excluídas considerando o ambiente de execução controlado das POCs neste trabalho. As métricas utilizadas foram Perfil do Risco Crítico e Top Tipos de Vulnerabilidade.

**Perfil do Risco Crítico:** as vulnerabilidades são categorizadas por uma ordem de criticidade. Para utilizar esta técnica, foi considerado o fator de Impacto Médio Ponderado, em inglês *Avg Weighted Impact*, das vulnerabilidades OWASP, que reflete o impacto das vulnerabilidades bem sucedidas.

As vulnerabilidades de maior criticidade e com maior incidência demandam uma análise mais minuciosa e solução prioritária.

**Top Tipos de Vulnerabilidade:** o propósito desta métrica é monitorar as vulnerabilidades mais frequentes, sendo um indicador eficaz alcançar o menor número possível de vulnerabilidades desprovidas de um plano de mitigação. Para empregar esta técnica, foi utilizado o fator de taxa média de incidência, em inglês "*Avg Incidence Rate*", das vulnerabilidades do OWASP, o qual reflete a média de incidência das vulnerabilidades.

De acordo com Akujobi (2021), outras métricas qualitativas são consideradas para uma avaliação abrangente do *pipeline* de segurança em um ambiente DevSecOps. Isso inclui a profundidade da análise dinâmica e estática, que avaliam quão minunciosas são as inspeções em tempo de execução por ferramentas dinâmicas e a capacidade de identificação

detalhada de vulnerabilidades antes do tempo de execução por ferramentas estáticas, respectivamente.

A métrica de intensidade dos ataques, neste trabalho considerada como métrica de risco crítico, é equiparada à métrica de risco crítico. A consolidação de conclusões direciona-se à efetiva utilização das conclusões geradas pelo *pipeline* de segurança (AKUJOBI, 2021). Este processo abrange a consolidação e a interpretação das descobertas, assim como a implementação de ações corretivas fundamentadas nessas conclusões. Para cada um destes aspectos, há um nível de maturidade, que pode ser baixo ou alto no *pipeline* (AKUJOBI, 2021). Portanto, foi utilizada a escala Likert para tratamento mais adequado de quão alto ou quão baixo o resultado se apresenta.

### 4.6.2 Análise de Resultados com Pesquisa-Ação

Além das métricas utilizadas na fase de análise e interpretação dos dados coletados, há a possibilidade de melhoria do *pipeline* conforme necessidade para identificar e mitigar vulnerabilidades ainda existentes por meio do ciclo de Pesquisa-Ação. As PoCs que possuírem o ciclo de Pesquisa-Ação, compreenderão as quatro etapas de análise de resultados, sendo uma adaptação do protocolo estabelecido em (GIL, 2017):

- **Coleta de dados:** nesta fase, foram coletados os dados de identificação da vulnerabilidade pelo *pipeline* para posterior análise.
- **Análise e interpretação dos dados:** com a coleta de dados, foi realizada a categorização, análise, discussão e interpretação destes dados coletados.
- **Elaboração do plano de ação:** nesta fase, foi elaborado um plano de ação para o problema, considerando os objetivos a serem atingidos e medidas de melhoria, identificando através do *pipeline* as vulnerabilidades existentes na aplicação e planejando como mitigá-las.
- **Divulgação dos resultados:** por fim, foi divulgado o resultado obtido nas fases anteriores.

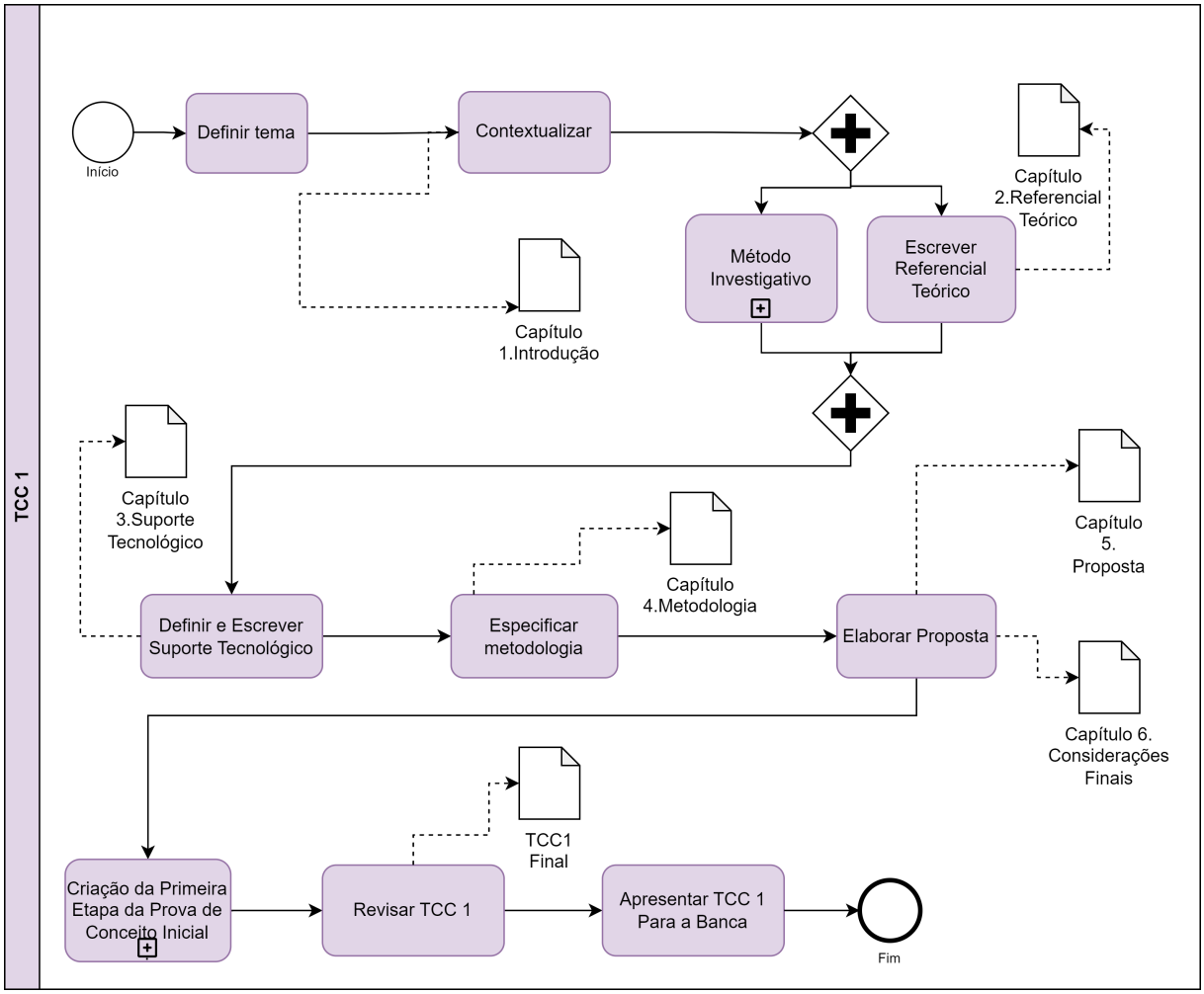
## 4.7 Fluxos de Atividades

Para a construção deste trabalho, foram elaborados fluxos BPMN (Business Process Model and Notation), apresentando as etapas do trabalho realizado nas fases de TCC 1 e TCC 2.

### 4.7.1 Atividades e Subprocessos - Primeira Etapa do TCC

A Figura 23 apresenta o fluxo de produção da primeira etapa deste Trabalho de Conclusão de Curso.

Figura 23 – Fluxo de atividades e subprocessos TCC1




Fonte: Autoria própria.

- **Definir tema**: Nesta primeira etapa, foi acordado o tema entre os autores e a orientadora, que foi delimitado na área de DevSecOps, compreendendo o título do trabalho..
  *Status*: Concluída.
  Insumo Gerado: DevSecOps: Um Estudo sobre a Aplicação de Segurança ao *Pipeline* DevOps de um Software, compreendendo o título do trabalho.

- **Contextualizar**: para compreender o contexto e o escopo do problema, foi realizada a etapa de contextualização da segurança no ciclo de vida de desenvolvimento de software.

*Status*: Concluída.
Insumos Gerado: Strings de Busca, Palavras-chave, dentre outros.

- **Método investigativo**: após a contextualização, foi feito o levantamento bibliográfico para embasamento teórico do trabalho.
  *Status*: Concluída.
  Insumos Gerados: Método Investigativo

- **Escrever Referencial Teórico**: a partir do método investigativo, elaborou-se o referencial teórico do trabalho.
  *Status*: Concluída.
  Insumo Gerado: Referencial Teórico

- **Definir e Escrever Suporte Tecnológico**: nesta etapa, os autores analisaram as ferramentas disponíveis na data da elaboração do trabalho e discutiram as opções que adequavam-se ao estudo proposto.
  *Status*: Concluída.
  Insumo Gerado: Suporte Tecnológico

- **Especificar Metodologia**: nesta atividade, foram definidos os métodos utilizados para a pesquisa científica.
  *Status*: Concluída.
  Insumo Gerado: Metodologia

- **Elaborar Proposta**: neste momento, foi elaborada a proposta do trabalho com base nas informações coletadas.
  *Status*: Concluída.
  Insumo Gerado: Capítulo. 5 - Proposta (Monografia da Etapa 1 do TCC)

- **Criação da Primeira Etapa da Prova de Conceito Inicial**: com o objetivo de comprovar a aplicação do tema, os autores implementaram a primeira etapa, constituída de uma aplicação com vulnerabilidades, da prova de conceito inicial.
  *Status*: Concluída.
  Insumo Gerado: Capítulo. 5 - Proposta (Monografia da Etapa 1 do TCC)

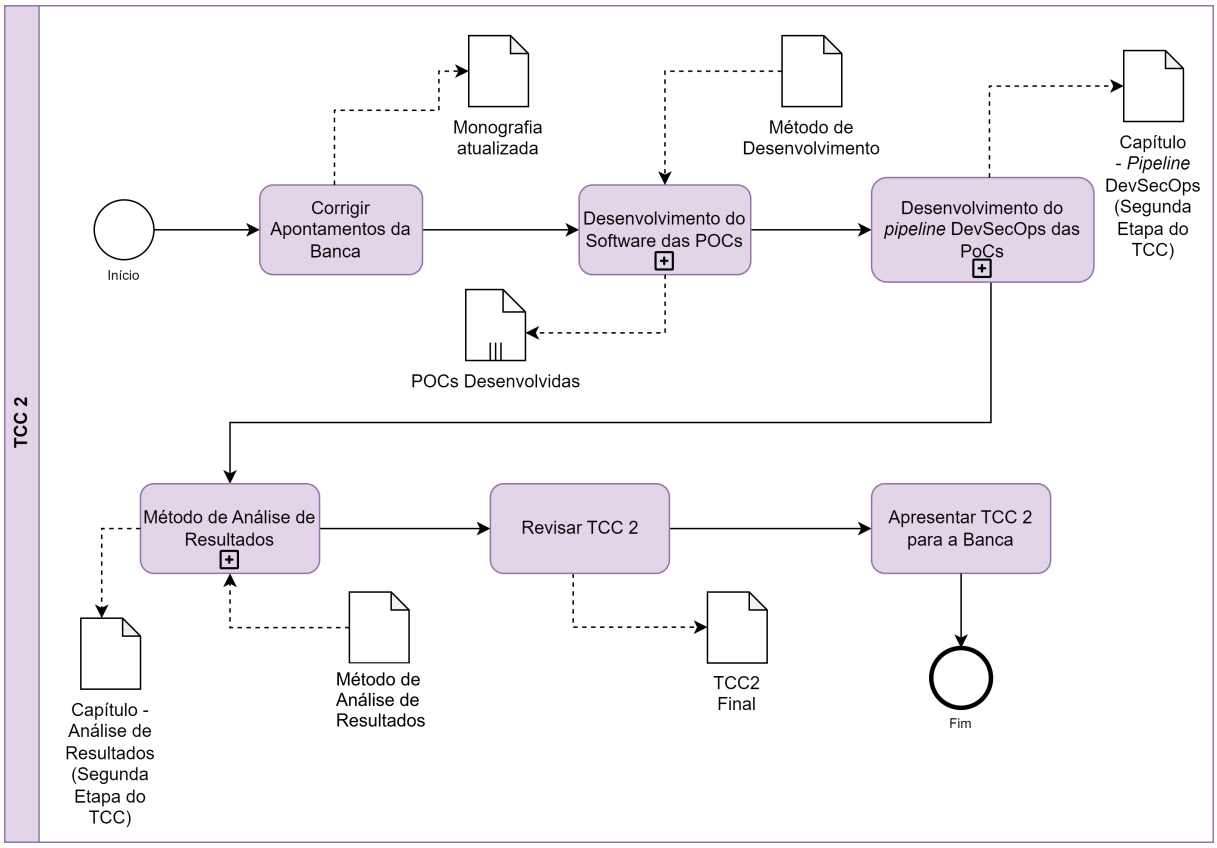
- **Revisar TCC 1**: para evitar erros de ortografia e desencadeamento do trabalho, foi realizada a revisão do TCC 1.
  *Status*: Concluída.

- **Apresentar o TCC 1 para a banca**: uma vez finalizada a revisão, o TCC 1 foi apresentado para a Banca definida.

### 4.7.2 Atividades e Subprocessos - Segunda Etapa do TCC

A Figura 23 apresenta o fluxo de produção da segunda etapa deste Trabalho de Conclusão de Curso.

Figura 24 – Fluxo de atividades e subprocessos TCC2




Fonte: Autoria própria.

- **Corrigir Apontamentos da Banca**: Antes de iniciar o desenvolvimento do TCC 2, foi definida uma etapa específica para correção de apontamentos feitos pela banca na apresentação do TCC 1.
  *Status*: Concluída.

- **Desenvolvimento do Software das PoCs**: Com as correções realizadas, os autores adentram na fase de desenvolvimento do software proposto nas PoCs.
  *Status*: Concluída.

- **Desenvolvimento do *pipeline* DevSecOps das PoCs**: Em paralelo com o desenvolvimento do software, foi desenvolvido também o *pipeline* DevSecOps da apli-

cação, aplicado desde o início do ciclo de desenvolvimento.
*Status*: Concluída.
Insumos Gerados: Pipeline DevSecOps.

- **Análise de Dados e Resultados**: Com a implementação da aplicação e a implantação do *pipeline*, foi realizada a análise dos dados e resultados obtidos.
  *Status*: Concluída.
  Insumos Gerados: Análise de Resultados.

- **Revisar TCC 2**: Com a análise dos resultados, o TCC 2 foi também revisado para evitar possíveis erros.
  *Status*: Concluída.
  Insumos Gerados: Presente Monografia.

- **Apresentar o TCC 2 para a banca**: Por fim, feita a revisão, o TCC deve ser apresentado para a Banca definida.
  *Status*: Concluída.

## 4.8 Cronograma de Atividades

A partir do fluxo de atividades elaborado, foram definidos os cronogramas de execução de atividades referentes às etapas inicial e final do TCC. O Quadro 2 apresenta o cronograma de trabalho da primeira etapa do TCC, e o Quadro 3 apresenta o cronograma de trabalho da segunda etapa do TCC.

Quadro 2 – Cronograma de atividades - primeira etapa do TCC

| Atividade/Subprocesso | Setembro | Outubro | Novembro | Dezembro |
|---|---|---|---|---|
| Definir tema | X | | | |
| Contextualizar | | X | | |
| Método Investigativo | | X | | |
| Escrever Referencial Teórico | | X | | |
| Definir e Escrever Suporte Tecnológico | | | X | |
| Especificar Metodologia | | | X | |
| Elaborar Proposta | | | X | |
| Criação da Primeira Etapa da Prova de Conceito Inicial | | | X | |
| Revisar TCC 1 | | | | X |
| Apresentar o TCC 1 para a banca | | | | X |

Fonte: Autoria própria.

Quadro 3 – Cronograma de atividades - segunda estapa do TCC

| Atividade/Subprocesso | Março | Abril | Maio | Junho | Julho |
|---|---|---|---|---|---|
| Corrigir Apontamentos da Banca | X | | | | |
| Desenvolvimento do Software das PoCs | | X | X | | |
| Desenvolvimento do *pipeline* DevSecOps das PoCs | | X | X | X | |
| Análise de Dados e Resultados | | X | X | X | |
| Revisar TCC 2 | | | | X | X |
| Apresentar o TCC 2 para a banca | | | | | X |

Fonte: Autoria própria.

## 4.9 Considerações Finais do Capítulo

Este capítulo detalhou a abordagem metodológica utilizada na elaboração deste projeto, classificando a pesquisa quanto aos critérios de abordagem, objetivos e procedimento. Além disso, detalhou o método investigativo, o método geral orientado a provas de conceito, o método de desenvolvimento e o método de análise de resultados. Por fim, foram acordados os fluxos de atividades e cronogramas da pesquisa, tanto da etapa inicial, quanto da etapa final.

Em resumo, a pesquisa é de abordagem Qualitativa/Quantitativa; de natureza Aplicada; com objetivos Exploratórios, e procedimentos estabelecidos via métodos: Provas de Conceito, para condução geral do *pipeline*; Scrum & XP & Kanban, para o desenvolvimento, e Pesquisa-Ação para análise de resultados. Em termos de métricas, os estudos concentram-se em: Eficácia do *Pipeline*, Perfil do Risco Crítico e Top Tipos de Vulnerabilidade.

# 5 Pipeline DevSecOps

## 5.1 Considerações Iniciais do Capítulo

O objetivo deste capítulo é realizar uma apresentação mais detalhada da aplicação do *pipeline* DevSecOps desenvolvido nas Provas de Conceito 1 e 2. Inicialmente, é apresentada a CONTEXTUALIZAÇÃO que embasou o presente trabalho. Sequencialmente, é apresentado o *pipeline* DevSecOps desenvolvido. Em seguida, há um descritivo sobre as PROVAS DE CONCEITO. Para cada Prova de Conceito, é apresentado o Objeto de Estudo, além das Vulnerabilidades exploráveis na aplicação, provendo insumos sobre cada vulnerabilidade, e apresentando o *pipeline* DevSecOps orientado às práticas e percepções adquiridas com o estudo exploratório das provas de conceito e suas vulnerabilidades. O capítulo termina com as CONSIDERAÇÕES FINAIS DO CAPÍTULO.

## 5.2 Contextualização

O aumento significativo no acesso e no uso da internet mudou a estratégia que grandes empresas possuiam em relação à oferta de serviços. Não demorou muito para que fosse necessário o desenvolvimento de plataformas *web* com a capacidade de atender a demanda de consumidores de todos os lugares do mundo. Essas lojas que, antes ofereciam seus produtos e serviços diretamente em suas lojas físicas, tiveram então uma expansão na área de atuação, agora, contando com o espaço conferido pela internet. Esse aumento pode ser comprovado quando analisados os dados de acesso à internet nos últimos anos. De 2011 até 2021, o acesso à internet no mundo mais do que dobrou, saindo de 32.7% em dezembro de 2011, para 66.2% em dezembro em 2021. A migração para esse ambiente virtual também favoreceu o surgimento de pequenos negócios (Internet World Stats, 2023).

Essa migração foi responsável por um aumento tanto na demanda quanto na oferta por soluções e serviços voltados para idealização, construção e manutenção de plataformas de venda *on-line*.

Não foram somente lojas que tiveram uma migração para o ambiente virtual. Diversas aplicações e soluções passaram a ter um sentido de existir considerando esse novo ambiente. A ascensão da internet trouxe motivos para o surgimento de redes sociais também.

Com todas essas plataformas, observou-se a necessidade em conferir uma maior segurança para os dados dos usuários. Plataformas e *sites* possuem os mais diferentes

objetivos. Uma plataforma pode focar em uma funcionalidade básica, que não demanda identificação alguma, mas também pode, como ocorre em um banco, oferecer diferentes operações aos seus usuários. Ambas aplicações têm o seu uso. Entretanto, a segunda possui uma necessidade de segurança muito maior.

A aplicação de mecanismos de segurança é muitas vezes tratada como algo de segundo plano por desenvolvedores e por empresas, visto o tempo demandado ao se aplicar medidas de segurança. Para uma empresa que desenvolve *software*, pode parecer atraente negligenciar aspectos de segurança dado que, muitas vezes, o cliente nem vai questionar sobre tais aspectos. A empresa prefere, assim, entregar o produto o mais rápido possível, agradando o cliente, ao invés de se aprofundar em detalhes técnicos de segurança.

Esse trabalho apresenta um estudo sobre a aplicação de segurança ao *pipeline* DevOps de um software, considerando insumos gerados por ferramentas de segurança, em aplicações desenvolvidas via provas de conceito; analisando as vulnerabilidades identificadas, e propondo formas de mitigá-las com base na literatura especializada. As ferramentas que compreendem o arcabouço tecnológico desse trabalho encontram-se no Capítulo 3 - Suporte Tecnológico. Já os demais detalhes conceituais de embasamento e metodológicos de condução do trabalho constam, respectivamente, no Capítulo 2 - Referencial Teórico e Capítulo 4 - Metodologia.

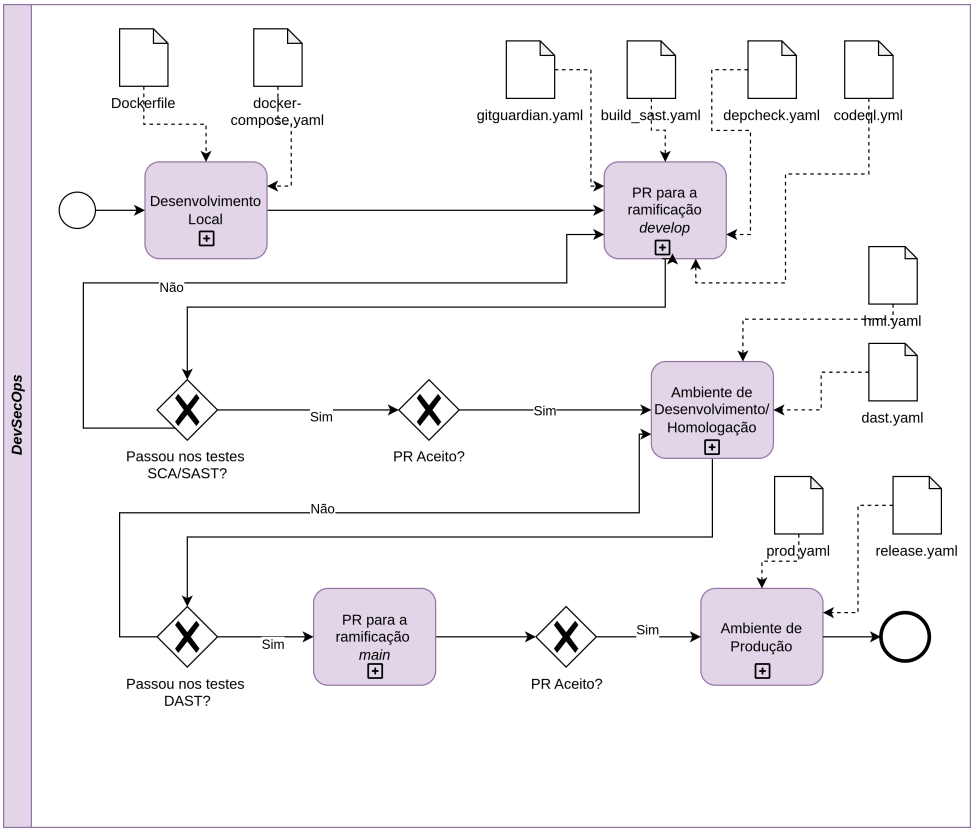
Seguem as provas de conceito desenvolvidas pelos autores, no intuito de esclarecer sobre como foram exploradas algumas típicas aplicações orientando-se por Segurança de Software.

As provas de conceito foram desenvolvidas pelos próprios autores e mantidas também por eles. Além disso, o *pipeline* foi desenvolvido e mantido pelos autores.

## 5.3 Pipeline DevSecOps

A Figura 25 apresenta o fluxo com as etapas do *pipeline* DevSecOps desenvolvido.

Figura 25 – Fluxo DevSecOps




Fonte: Autores

O fluxo compreende os estágios de desenvolvimento local, de abertura de *Pull Requests* no Github, da aprovação do *Pull Request* para o ambiente de homologação até a geração de nova *release* em produção. O ambiente de desenvolvimento/homologação e o ambiente de produção considerados na infraestrutura do *pipeline* encontram-se nas ramificações (*branches*) *develop* e *main* do Github, respectivamente.

Os desenvolvedores utilizaram o ambiente de desenvolvimento/homologação para manter a versão do código com as mesclas (*merges*), para homologar as novas *features* e para realizar testes de sistema e testes dinâmicos (DAST). O ambiente de produção integra a versão mais atualizada e estável do código. A cada mescla (*merge*) realizada no ambiente de produção, foi gerada uma nova *release* da aplicação. A *action* criada a partir do arquivo prod.yaml executa os jobs e realiza o *Deploy* Contínuo. Nesse ponto, o ambiente já encontrava-se pronto para ser utilizado pelo usuário com as novas funcionalidades.

**Desenvolvimento Local:** para que, ao final do fluxo, o sistema fosse entregue ao usuário, a etapa inicial foi o desenvolvimento local. Com a finalidade de garantir que durante a etapa do Desenvolvimento Local os desenvolvedores utilizariam versões equivalentes das dependências, foi adotado o *Docker* e foram criados os arquivos *Dockerfile.yml* e *docker-compose.yml*. O Código 13 exemplifica um dos arquivos *docker-compose.yml*.

Código 1 – *Docker-compose.yml* do *Frontend* da POC1

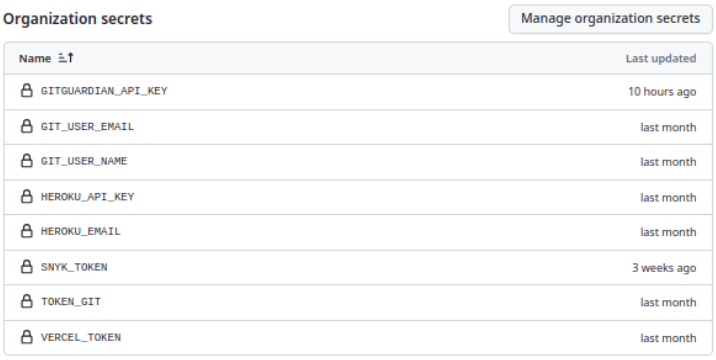
```
services:
  tcc-password-manager:
    container_name: frontend_password_manager
    build: .
    ports:
      - 3000:3000
    environment:
      PORT: 3000
    volumes:
      - .:/app
      - /app/node_modules
    networks:
      - frontend_password_manager

volumes:
  node_modules:

networks:
  frontend_password_manager:
```

Fonte: Autoria própria.

Este arquivo, juntamente com o *Dockerfile*, define as versões, serviços e as portas onde a aplicação e o banco de dados, no caso do *Backend*, irão rodar.

**Segurança no versionamento de código:** Após a etapa de Desenvolvimento Local, o compartilhamento e o versionamento do código ocorreram no *Github*. O Github possui funcionalidades disponíveis para códigos públicos que colaboram com a segurança das aplicações. Assim, foram estabelecidos procedimentos de segurança nos repositórios, como a utilização da funcionalidade *secrets* para guardar os segredos. As *secrets* foram separadas em *secrets* da organização (vide Figura 26), *secrets* do repositório *Frontend* (vide Figura 27) e *secrets* do repositório *Backend* (vide Figura 28).

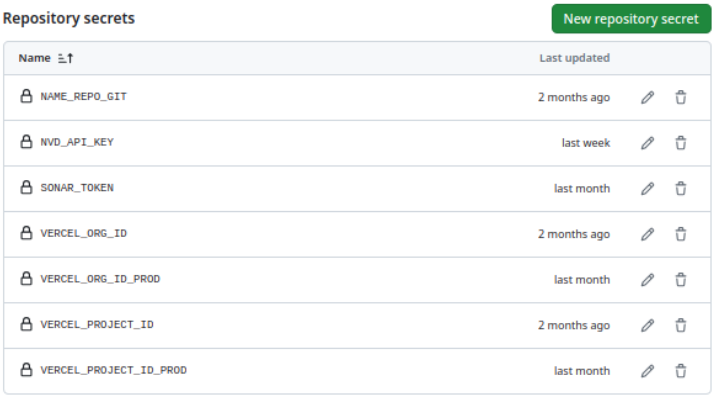
Figura 26 – *Secrets* da organização




Fonte: Autores

As *secrets* da organização são os segredos compartilhados por todos os repositórios. Estas *secrets* são aquelas relacionadas ao git como `GIT_USER_EMAIL`, `GIT_USER_NAME` e as *secrets* de autenticação em ferramentas como `HEROKU_API_KEY`, `HEROKU_EMAIL`, `SNYK_TOKEN`, `TOKEN_GIT` e `VERCEL_TOKEN`. Para cada repositório *Frontend* da organização, foram estabelecidas as *secrets* da Figura 27.

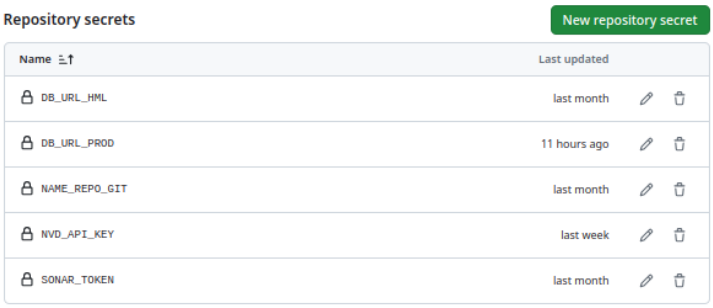
Figura 27 – *Secrets Frontend*




Fonte: Autores

As *secrets* dos repositórios de *Frontend* são aquelas específicas para cada repositório, como por exemplo, o `NAME_REPO_GIT`, `NVD_API_KEY`, `SONAR_TOKEN`. As *secrets* referentes ao *Deploy* no Vercel também foram definidas para cada repositório, sendo elas: `VERCEL_ORG_ID`, `VERCEL_ORG_ID_PROD`, `VERCEL_PROJECT_ID`, `VERCEL_PROJECT_ID_PROD`. Já no *Backend*, as *secrets* estão contidas, conforme apresentado na Figura 28.

Figura 28 – *Secrets Backend*

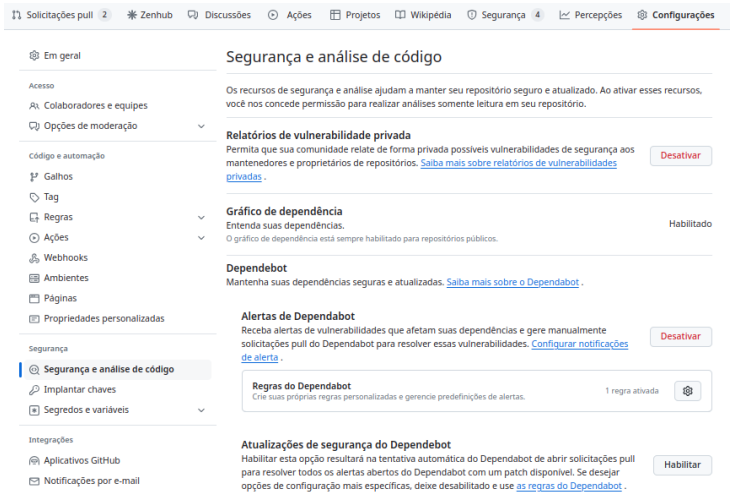



Fonte: Autores

As *secrets* utilizadas nos repositórios de *Backend*, além das que são específicas do repositório, foram as *secrets* direcionadas ao *Deploy* no Heroku como `API_SECRET`, `DB_URL`, `DB_URL_PROD`, `NAME_REPO_GIT`, `NVD_API_KEY`, `SONAR_TOKEN`.

Além das *secrets*, outro procedimento de segurança efetuado no *Gihub* foi alterar configurações de segurança e análise de código, conforme consta na Figura 29

Figura 29 – Configurações de segurança do Github

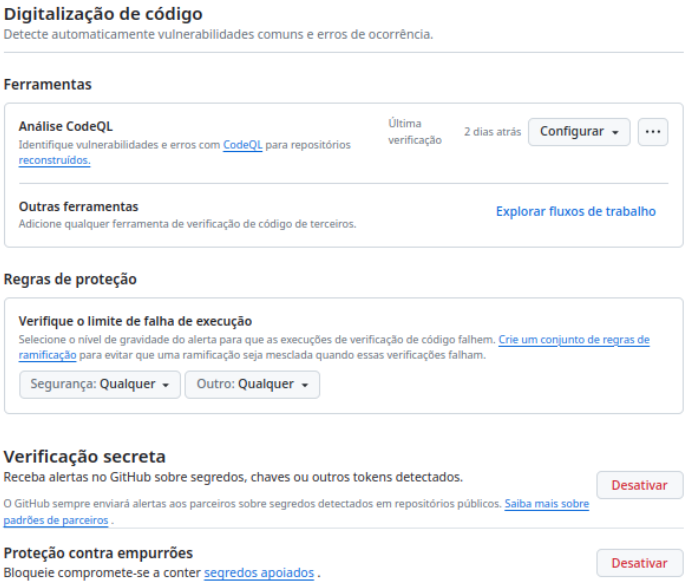



Fonte: Autores

Nas configurações, os relatórios de vulnerabilidade privada foram habilitados para que a comunidade acione os mantenedores sobre as falhas de segurança detectadas. O gráfico de dependências encontra-se ligado por padrão, apresentando as dependências já classificadas por suas vulnerabilidades. Adicionalmente, foram ativados os alertas do *dependabot*.

Ademais, a Figura 30 demonstra parametrizações de funcionalidades do *Github Advanced Security* que também passaram por alterações, como o CodeQL CLI, o *CodeScanning* e o *SecretScanning*.

Figura 30 – Configurações avançadas de segurança no Github

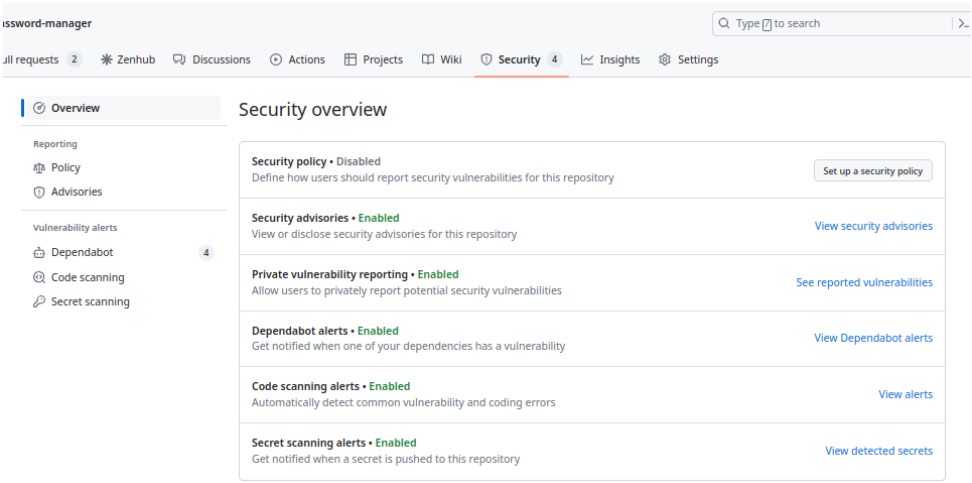



Fonte: Autores

O CodeQL CLI viabiliza as configurações avançadas do CodeQL nos *workflows* do *Github*. O CodeScanning analisa o código e verifica vulnerabilidades de segurança acionando alertas no repositório. Com o *SecretScanning*, o *Github* verifica os repositórios em busca de tipos de segredo conhecidos e gera alertas ao encontrar. Quando as vulnerabilidades são corrigidas, os alertas são automaticamente encerrados.

Ainda no Github, para acesso aos alertas e avisos gerados, foi habilitada a aba de segurança, como demonstra a Figura 30.

Figura 31 – Aba de Segurança no Github




Fonte: Autores

A aba de segurança reúne as informações de segurança em um único local, facilitando o monitoramento, o controle e a organização deste aspecto no repositório. A apresentação de alertas de segurança na aba *security* do *Github* suporta arquivos SARIF (*Static Analysis Results Interchange Format*).

**Fluxo de trabalho de versionamento:** O *pipeline* propõe um processo de versionamento em que os desenvolvedores efetuam (*commits*) em uma ramificação (*branch*) e abrem um *Pull Request* para a ramificação de desenvolvimento/homologação, solicitando a revisão por pares.

**Pull Request para Develop/Homologação:** Ao abrir o PR, as *actions codeql.yaml*, *build_sast.yaml*, *depcheck.yaml* e *gitguardian.yaml* são acionadas realizando as etapas de Build, SCA e os Testes Estáticos (SAST). A etapa de configurações do fluxo de trabalho (*workflow*) de Build e SAST presente no "*build_sast.yaml*"encontra-se no Código 2.

Código 2 – Configurações do arquivo `build_sast.yaml`

```yaml
name: Build and SAST

on:
  pull_request:
    branches: [develop]

jobs:
  build_sast:
    name: 'build_sast'
    runs-on: ubuntu-latest

    steps:
      - name: Checkout Repository
        uses: actions/checkout@v4

      - name: Config GitHub
        run: |
          git config --global user.email "${{secrets.GIT_USER_EMAIL}}"
          git config --global user.name "${{secrets.GIT_USER_NAME}}"
```

Fonte: Autoria própria.

Para o Build, foi definido o sistema operacional Ubuntu em sua versão mais recente (*latest*). A configuração do usuário para a autorização de interação com os recursos do *Github* foi feita com as *Secrets*, mantendo a segurança e o sigilo das informações. A etapa de auditoria está evidenciada no Código 3.

Código 3 – Auditoria

```
- name: Run npm audit
  uses: oke-py/npm-audit-action@v2
  with:
    audit_level: low
    create_issues: true
    github_token: ${{ secrets.TOKEN_GIT }}
    issue_labels: vulnerability
```

Fonte: Autoria própria.

Na etapa (*step*) de auditoria, o comando *npm-audit* é empregado para analisar o arquivo *package-lock.json*, retornando o relatório das vulnerabilidades encontradas e a gravidade de cada uma delas. Este relatório é exibido como um comentário no *Pull Request* facilitando a visualização para os desenvolvedores. O Código 4 mostra a instalação das dependências pela *action*.

Código 4 – Instalação de dependências

```
- name: Install npm ci
  env:
    NODE_AUTH_TOKEN: ${{ secrets.TOKEN_GIT }}
  run: npm ci
```

Fonte: Autoria própria.

As dependências são então instaladas com o *npm-ci*. A adoção do comando *npm-ci*, em detrimento do *npm install*, baseia-se na capacidade do *npm-ci* instalar as dependências como especificado no arquivo *package-lock.json*, e não no *package.json*, como faz o comando *npm install*. Este comportamento otimiza a eficiência do *pipeline*.

Durante a análise de dependências, para a análise da composição do software (SCA), foram implementadas as ferramentas *Snyk*, *OWASP Dependency Check* e *Dependabot*. O arquivo *depcheck.yaml* inicia-se com a ferramenta Snyk do Código 5 após as configurações.

Código 5 – *Action* Snyk

```
1   snyk:
2     name: 'snyk'
3     ...
4
5     steps:
6         ...
7
8       - name: Run Snyk to check for vulnerabilities
9         uses: snyk/actions/node@master
10        continue-on-error: true
11        env:
12          SNYK_TOKEN: ${{ secrets.SNYK_TOKEN }}
13        with:
14          args: --sarif-file-output=snyk.sarif
```

Fonte: Autoria própria.

O *workflow* utiliza a *action* fornecida pela Snyk considerando a versão mais recente da *master*. A declaração *continue-on-error: true* permite que o fluxo de trabalho permaneça em execução mesmo se a ferramenta encontrar vulnerabilidades. Esta declaração possibilita a geração dos relatórios em formato de arquivo SARIF. Introduz-se, então, a análise do OWASP *Dependency Check* (vide Código 9).

Código 6 – *Action* OWASP *Dependency Check*

```
1   owasp_depcheck:
2     name: 'owasp_depcheck'
3     ...
4
5     steps:
6         ...
7
8       - name: Depcheck
9         uses: dependency-check/Dependency-Check_Action@main
10        id: Depcheck
11        with:
12          project: 'tcc-password-manager'
13          path: './'
14          format: 'ALL'
15          out: 'depcheck'
16          args: >
17            --enableRetired
18            --nvdApiKey ${{ secrets.NVD_API_KEY }}
```

Fonte: Autoria própria.

A ferramenta OWASP *Dependency Check* também verifica as dependências vulneráveis. A *action* é especificada em sua versão mais recente da *master* e os parâmetros fornecidos são o nome do projeto, o caminho e o formato do relatório (ALL). O Dependabot é um recurso do *Github*, e também foi aplicado na verificação de dependências não seguras. Os avisos do Dependabot podem ser acessados no setor *security* do Github.

Com isso, os testes estáticos iniciam-se com a *action* do SonarCloud do Código 7.

Código 7 – *Step* do SonarCloud

```
1       - name: SonarCloud
2         uses: actions/checkout@v3
3         with:
4           fetch-depth: 0
5
6       - name: SonarCloud Scan
7         if: always()
8         uses: SonarSource/sonarcloud-github-action@master
9         env:
10          GITHUB_TOKEN: ${{ secrets.TOKEN_GIT }}
11          SONAR_TOKEN: ${{ secrets.SONAR_TOKEN }}
```

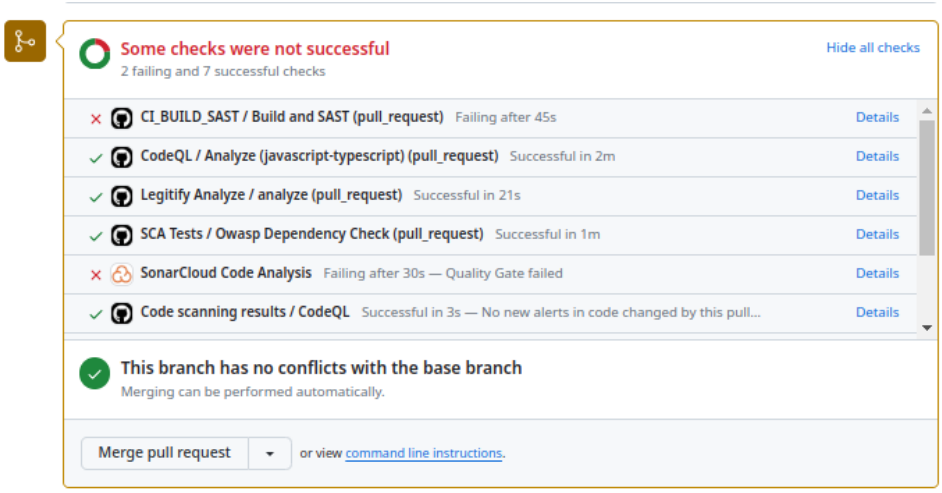
Fonte: Autoria própria.

Na etapa de SAST, o SonarQube, em sua versão Cloud, realizou a verificação da qualidade do código-fonte. O código do PR precisa atender ao Portão de Qualidade para que a etapa seja bem-sucedida. Como o contexto deste trabalho é centralizado em segurança de software, foi definido o Portão de Qualidade *Quality Gate* sendo zero pontos de acesso de segurança (*security hotspots*). O *bot* do *SonarCloud* comenta no *Pull Request* quando o teste estático foi concluído com sucesso e quando falhou, permitindo visualizar detalhes como caminho, linha e motivo da falha do portão de qualidade.

Por fim, o CodeQL foi aplicado na execução de análises de segurança do código e na produção dos alertas no Github.

Após a execução das *actions*, o *Github* exibe de forma detalhada as ações com sucesso e falhas (vide Figura 32).

Figura 32 – Exemplo de *workflows* bem sucedidos e com falha no GitHub




Fonte: Autores

Mesmo na ausência de falhas na execução dos *workflows*, o desenvolvedor deve navegar até o painel *Security* para examinar se há outras vulnerabilidades a serem mitigadas.

**Relatórios de SCA/Análises Estáticas:** Os relatórios de auditoria e dependências vulneráveis são salvos no formato SARIF, para serem apresentados na aba *security* e são também guardados como artefatos do *workflow* permitindo o *download*. Estes processos estão no Código 8.

Código 8 – Código *Upload* SARIF e Artefatos

```
1       - name: Upload SARIF results file
2         uses: github/codeql-action/upload-sarif@v3
3         with:
4           sarif_file: depcheck/dependency-check-report.sarif
5
6       - name: Upload Test results
7         uses: actions/upload-artifact@v4
8         with:
9             name: Depcheck report
10            path:  ${{github.workspace}}/depcheck
```

Fonte: Autoria própria.

O desenvolvedor procede com correção das vulnerabilidades identificadas pelas ferramentas.

Com a correção destas vulnerabilidades, o desenvolvedor seleciona um revisor para o seu código. Neste momento, é empregado o método de inspeção de código por meio de

revisão por pares. Durante a inspeção, o revisor, embasado pela OWASP Top 10, adiciona comentários no *Pull Request* com Relatórios de Inspeção de Vulnerabilidades. O conteúdo destes comentários é composto pela Vulnerabilidade OWASP Top 10, a Descrição, o Impacto e uma sugestão de correção. O revisor deve realizar também Testes de Sistema que permitem perceber comportamentos inesperados e vulneráveis no *Software*.

Posteriormente, com os ajustes solicitados em todos os relatórios, o PR pode ser combinado/mesclado (em inglês, *merged*) ao ambiente de homologação.

**Homologação:** Depois da aprovação do PR e mescla para a ramificação de homologação, é feito o *Deploy* Contínuo atualizando o ambiente de homologação com as modificações inseridas, mediante o uso do Vercel no *Frontend* e do Heroku no *Backend*.

Para a etapa de DAST funcionar, as fases anteriores devem ser concluídas com sucesso, já que as ferramentas DAST utilizam a URL da aplicação para testes ativos a partir de requisições.

Dois tipos de *scanner* Zed Attack Proxy (ZAP) foram adicionados no *pipeline*: a varredura de linha de base (*baseline scan*) e a verificação completa (*full scan*). A *baseline scan* executa o *spider* na URL, simulando o comportamento do usuário navegando na aplicação, registrando as páginas acessíveis e realizando testes passivos. A verificação completa (*full scan*) realiza a varredura *ajax* e os testes ativos. O Código 9 traz o código *dast.yaml* do *step* do *Baseline Scan* do ZAP.

Código 9 – Código do *Baseline Scan* ZAP

```
1   zap_baseline:
2     name: 'zap_baseline_scan'
3     runs-on: ubuntu-latest
4     needs: config_git
5
6     steps:
7       - name: Checkout
8         uses: actions/checkout@v4
9         with:
10          ref: develop
11        ...
12
13      - name: ZAP Baseline Scan
14        uses: zaproxy/action-baseline@v0.12.0
15        with:
16          token: ${{ secrets.TOKEN_GIT }}
17          docker_name: 'ghcr.io/zaproxy/zaproxy:stable'
18          target: 'https://hml-tcc-password-manager.vercel.app'
19          cmd_options: >
20            - a
21            - j
22          allow_issue_writing: 'true'
23          artifact_name: 'zap_baseline_scan'
24        ...
```

Fonte: Autoria própria.

A *action* utilizada é mantida pelos mantenedores do ZAP e inclui a imagem *Docker* da ferramenta, além do *target*, que é a URL da aplicação e os parâmetros passados no *cmd_options*. O parâmetro -a indica que são incluídas as regras de *scan* ativas e passivas. O parâmetro -j sinaliza a varredura opcional *ajax*. Como no *Baseline Scan*, é feito também o *Full Scan* no arquivo *dast.yaml*, segundo o Código 10.

Código 10 – Código do *Full Scan* ZAP

```yaml
zap_full_scan:
  name: 'zap_full_scan'
  ...
  steps:
      ...
    - name: ZAP Full Scan
      uses: zaproxy/action-full-scan@v0.10.0
      with:
        token: ${{ secrets.TOKEN_GIT }}
        docker_name: 'ghcr.io/zaproxy/zaproxy:stable'
        target: 'https://hml-tcc-password-manager.vercel.app'
        cmd_options: >
          - j
          - a
        allow_issue_writing: 'true'
        artifact_name: 'zap_full_scan'
      ...
```

Fonte: Autoria própria.

As configurações do *job* do *full scan* são semelhantes às do *baseline scan*, alterando apenas o tipo de varredura realizada. Para auxiliar nestes testes dinâmicos, a *action* do Nikto consta também como um *job* do arquivo *dast.yaml* (vide Código 11)

Código 11 – Código Nikto

```yaml
nikto:
  name: 'nikto'
  ...
    - name: Scan with nikto
      uses: thereisnotime/action-nikto@master
      with:
        url: "https://hml-tcc-password-manager.vercel.app/"
        additional_args: "-Option FAILURES=0 -o $FILE_PATH -Format json"
      continue-on-error: true
```

Fonte: Autoria própria.

O Nikto operou no *scanner* do servidor web em busca de arquivos e configurações vulneráveis. Para a *action*, foi passada a URL do *Deploy* de homologação e também os parâmetros "additional_args", sendo o -Option FAILURES=0 responsável por determinar que não há quantidade máxima de falhas definidas. Ou seja, o *scanner* não deve ser interrompido enquanto não finalizar todas as inspeções. O -o $FILE_PATH -Format json define o json como formato de saída do relatório.

**Relatórios dos Testes Dinâmicos:** Os relatórios de DAST são salvos em formato *json* pasta *reports* da ramificação *develop* em cada repositório, como no Código 14, visto que o *trigger* das ferramentas DAST é o PR fechado na *Develop*.

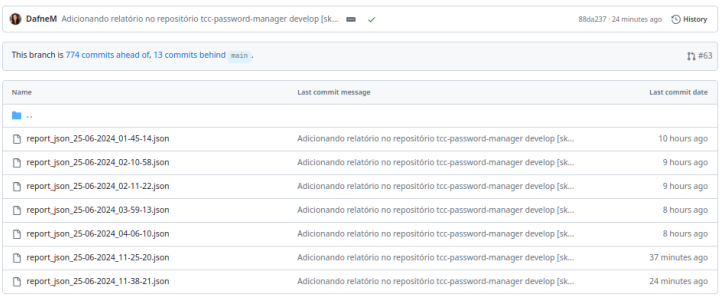
Código 12 – Código para salvar relatórios no *Github*

```
- name: Save ZAP Full Scan Reports
    if: always()
    env:
      TOKEN_GIT: ${{ secrets.TOKEN_GIT }}
      NAME_REPO_GIT: ${{ secrets.NAME_REPO_GIT }}
      TOOL_NAME: zap_full_scan
      REPORT_FORMAT: json
      REPORT_NAME: report_json
    run: |
      cd scripts
      chmod +x save_report.sh
      ./save_report.sh $GITHUB_WORKSPACE "${{github.event.repository.name }}"
      "${{ github.ref_name }}"
```

Fonte: Autoria própria.

Estes relatórios gerados pela *action dast.yaml* são salvos para registrar um histórico, facilitar a rastreabilidade, e permitir auditorias tardias (vide Figura 33).

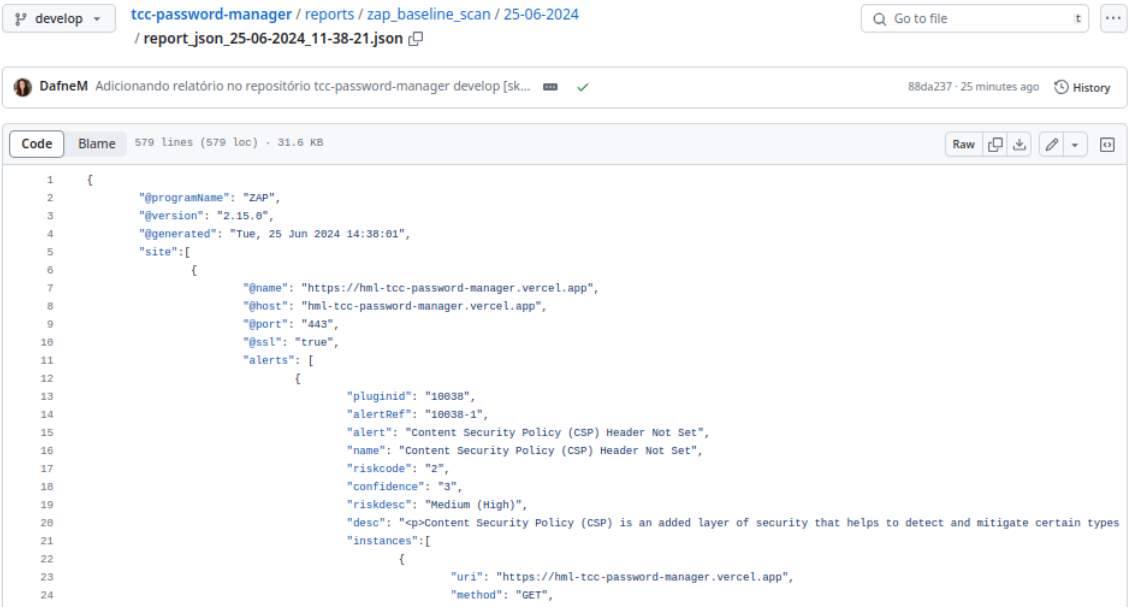
Figura 33 – Pasta com data de relatório




Fonte: Autores

São, portanto, criadas pastas cujos nomes são a data da realização dos testes. Dentro dessas pastas, são mantidos os arquivos com o registro dos horários dos testes realizados. Na Figura 34, há um exemplo de cada arquivo de relatório gravado com data e hora.

Figura 34 – Exemplo de relatório *.json* produzido pelas ferramentas DAST




Fonte: Autores

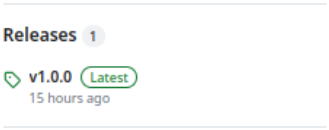
Os arquivos são armazenados exclusivamente no formato JSON para preservar a integridade do armazenamento no *GitHub*.

Com a correção dos alertas explicitados nos relatórios dos testes dinâmicos, podem ser realizados novos testes de sistema. Após a conclusão satisfatória dos testes de sistema, abre-se um PR para a *main*/produção, que deve ser analisado e pode ser mesclado na *main*.

**Produção:** Na última etapa, a partir do *Pull Request* aceito e mesclado, é feito o *Deploy* Contínuo no ambiente de produção para a utilização da aplicação pelo usuário final. Além disso, a *action release.yaml* é acionada e uma nova *release* é gerada.

A *action release.yaml* aciona o *script release.js*. A *release* é baseada na etiqueta (*LABEL*) do PR, podendo ser MAJOR RELEASE, MINOR RELEASE ou PATCH RELEASE, conforme o versionamento semântico, no formato vX.Y.Z, onde X representa a *release major*, com mudanças significativas de novas funcionalidades; Y representa uma *release minor*, que incrementa funcionalidades que são compatíveis com versões anteriores da aplicação; e Z, *patch*, que representa uma correção de vulnerabilidade ou *bug*. Com a geração da nova *release*, os usuários finais podem acessar o *software* nas versões mais recentes disponibilizadas. A Figura 35 mostra um exemplo de primeira *Release Major* no repositório.

Figura 35 – Exemplo de *Release*




Fonte: Autores

Uma vez definidos e aprofundados todos os estágios do *Pipeline*, os autores aplicaram o *Pipeline* DevSecOps nas PIPELINE DEVSECOPS especificadas.

## 5.4 Provas de Conceito

Há algo em comum nas aplicações escolhidas pelos autores, sendo o fato de ambas necessitarem manipular e controlar dados sensíveis dos usuários. Sendo assim, são aplicações muito prejudicadas, se sofrerem algum ataque de um invasor. Portanto, são consideradas aplicações críticas (JUNIOR, 2003).

Diante das colocações expostas anteriormente, optou-se por duas aplicações em particular, sendo: um gerenciador de senhas e uma aplicação bancária.

### 5.4.1 POC 1 - Gerenciador de Senhas

O Método Orientado a Provas de Conceito, estabelecido no Capítulo 4 - Metodologia, descreve as etapas que cada prova de conceito deve conter.

A primeira aplicação escolhida pelos autores foi a de um gerenciador de senha. Como já mencionado anteriormente na CONTEXTUALIZAÇÃO, um aumento no acesso à *internet* foi responsável pelo surgimento de plataformas oferecendo os mais diferentes serviços, que muitas vezes limitam as funcionalidades para usuários cadastrados. Nesse sentido, seguem as etapas, para o caso da primeira POC.

#### 5.4.1.1 Definição da PoC

Organizada em Requisitos da PoC, Sistema Objeto de Estudo e Vulnerabilidades OWASP Top 10.

**Requisitos da PoC**

Os requisitos funcionais definidos para a aplicação vulnerável desta PoC foram:

- O usuário deve conseguir se cadastrar para acessar a aplicação;

- O usuário deve realizar acesso com *login* utilizando senha;
- O usuário deve conseguir acessar uma lista com as credenciais previamente criadas;
- O usuário deve conseguir cadastrar novas credenciais, e
- O usuário deve conseguir remover credenciais previamente cadastradas.

#### Sistema Objeto de Estudo - Gerenciador de Senhas

As mais diferentes senhas são um problema para os usuários, que muitas vezes podem esquecer a senha que usaram para uma determinada plataforma. Um gerenciador de senha tem assim sua utilidade, pois se propõem a guardar todas as senhas do usuário, e, com somente uma única, o usuário pode ter acesso a todas as demais. Tal aplicação também pode ser considerada crítica, considerando o que ela se propõem a fazer: guardar todas as senhas do usuário, assim, o acesso por uma terceira pessoa configura um grande risco.

A aplicação contendo vulnerabilidades pode ser acessada pelos repositórios, na ramificação *results*:

- Gerenciador de Senhas: https://github.com/tcc-lucas-dafne/tcc-password-manager
- Gerenciador de Senhas API: https://github.com/tcc-lucas-dafne/tcc-password-manager-api

Para acesso à aplicação, o usuário deve primeiro realizar o cadastro (vide Figura 36) e, posteriormente, o *login* na plataforma (vide Figura 37). Com a conta criada, o usuário passa a ter acesso a uma *header* para navegação. Pela *header*, vide Figura 38, o usuário pode cadastrar novas credenciais; fazer o *logout* da aplicação, e visualizar/editar o próprio perfil. A tela principal da aplicação possui uma lista de credenciais cadastradas pelo usuário. Por padrão, as senhas são ocultadas. Também é possível remover as credenciais por meio de um ícone de lixeira.

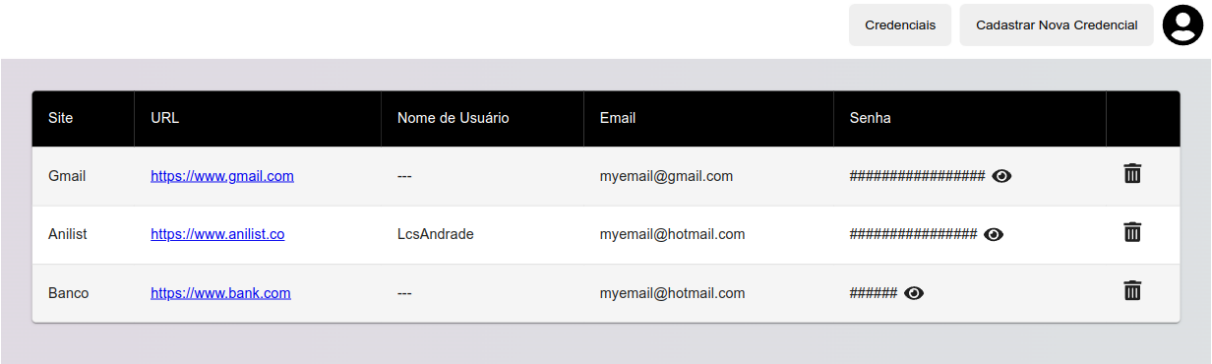
Figura 36 – Página de Cadastro da Aplicação de Gerenciamento de Senhas


Fonte: Autores

Figura 37 – Página de *Login* da Aplicação de Gerenciamento de Senhas


Fonte: Autores

Figura 38 – Página de Listagem das Credenciais da Aplicação de Gerenciamento de Senhas




Fonte: Autores

### Vulnerabilidades OWASP Top10 em Estudo

As vulnerabilidades Top 10 OWASP implantadas nesta aplicação foram A01, A02, A03, A04, A05, A06, A07 e A09.

### A01 Quebra de Controle de Acesso:

Após realizar o *login*, o usuário tem acesso à tela de listagem de credenciais, contendo dados de cada credencial cadastrada, conforme Figura 38. Por meio dela, o usuário consegue visualizar cada credencial cadastrada. A vulnerabilidade presente nesta página permite que um segundo usuário tenha acesso às credenciais de outro usuário. Isso ocorre porque a listagem utiliza um identificador (ID) único e facilmente identificável, presente na *URL* da página, para carregar as credenciais do usuário. Por exemplo, a listagem do usuário com ID 1 seria https://localhost:3000/1, enquanto que para o usuário com ID 2 seria http://localhost:3000/2. Com uma simples alteração na URL, o usuário de ID 1 consegue acessar as credenciais do usuário 2. Essa vulnerabilidade é descrita pelo A01 - Quebra de Controle de Acesso.

A aplicação possui outras vulnerabilidades que podem ser exploradas em todas as páginas: o token JWT (*JSON Web Token*) retornado no *login* do usuário não tem sua expiração verificada ao realizar requisições, permitindo assim que o usuário tenha acesso indefinido à conta. Essa vulnerabilidade é representada pelo A01 - Quebra de Controle de Acesso.

A aplicação também tem outras duas vulnerabilidades A01. Primeiro, a POC1 não possui um *middleware* de autenticação. Segundo, não possui limite máximo estabelecido para chamadas na API.

### A02 Falhas Criptográficas:

Esta vulnerabilidade pode ser percebida em dois momentos na aplicação, no armazenamento das senhas do usuário, que é feito em texto claro, e na ausência de validação de assinatura do *token*.

### A03 Injeção:

Pela tela de *login*, conforme consta na Figura 37, onde o usuário informa seu email e senha para acesso à plataforma, é possível explorar a vulnerabilidade de SQL *Injection*, referenciada no OWASP Top Ten como A03 - Injeção. Com essa vulnerabilidade, um usuário mal-intencionado pode injetar comandos SQL com o objetivo de realizar operações não permitidas no banco de dados da aplicação. No caso desta aplicação, como exemplo de uso, o usuário pode utilizar os caracteres '– no campo de email, após informar um email válido, e assim conseguir realizar o *login*, mesmo sem fornecer a senha de acesso.

### A04 Design Inseguro

Alguns tipos de dados não são definidos na aplicação, usando a diretiva *any*. Essa falta de definição possibilita ao invasor adicionar estruturas de dados inseguras na POC.

### A05 Configuração Incorreta de Segurança

Esta vulnerabilidade foi adicionada, pois não há configuração de CORS (*Cross-Origin Resource Sharing)* alguma permitindo que os recursos sejam acessados por diferentes origens e facilitando os ataques de segurança.

### A06 Componentes Vulneráveis e Desatualizados:

A aplicação também possui componentes vulneráveis. O componente vulnerável propositalmente determinado pelos autores foi o *Bootstrap* em versão <=3.4.0. Esses componentes são identificados pelo próprio *npm* (*Node Package Manager*) durante o processo de instalação dos pacotes.

**A07 Falhas de Identificação e Autenticação:**

A tela de cadastro, conforme consta na Figura 36, onde o usuário cria sua conta na aplicação, apresenta vulnerabilidades. Por exemplo, a ausência de requisitos mínimos para a senha do usuário, permitindo a criação de senhas muito básicas.

**A09 Falhas de registro e monitoramento de segurança:**

Não foram adicionados *logs* com as ações realizadas na aplicação.

#### 5.4.1.2 Solução

Organizada em Aplicação do Pipeline, Resolução de Vulnerabilidades e Análise de Resultados.

**Aplicação do *Pipeline***

Para a aplicação na Prova de Conceito 1 do *Pipeline* DevSecOps desenvolvido e a análise dos resultados, foram adicionados os arquivos de fluxos de trabalho (*workflows*); e criadas as ramificações *results* tanto no *Frontend* quanto no *Backend* do *Software*. Nestas ramificações, os desenvolvedores implantaram propositalmente todas as vulnerabilidades explicitadas na Seção 5.4.1.1. Em seguida, os desenvolvedores abriram um *Pull Request* (PR) para a ramificação *develop* para engatilhar as análises estáticas.

**Aditoria**

npm-audit

A primeira análise realizada durante o *Build* gerou o relatório auditoria a partir do comando *npm-audit* no repositório de *Frontend* (vide Figura 39).

Figura 39 – Componentes vulneráveis identificados pelo *npm* no *Frontend* da POC1

```
# npm audit report

bootstrap  <=3.4.0
Severity: moderate
bootstrap Cross-site Scripting vulnerability - https://github.com/advisories/GHSA-ph58-4vrj-w6hr
XSS vulnerability that affects bootstrap - https://github.com/advisories/GHSA-3mgp-fx93-9xv5
Bootstrap Vulnerable to Cross-Site Scripting - https://github.com/advisories/GHSA-9v3m-8fp8-mj99
Bootstrap Cross-site Scripting vulnerability - https://github.com/advisories/GHSA-4p24-vmcr-4gqj
Bootstrap vulnerable to Cross-Site Scripting (XSS) - https://github.com/advisories/GHSA-3wqf-4x89-9g79
Bootstrap Cross-site Scripting vulnerability - https://github.com/advisories/GHSA-7mvr-5x2g-wfc8
fix available via `npm audit fix --force`
Will install bootstrap@3.4.1, which is outside the stated dependency range
node_modules/bootstrap

nth-check  <2.0.1
Severity: high
Inefficient Regular Expression Complexity in nth-check - https://github.com/advisories/GHSA-rp65-9cf3-cjxr
fix available via `npm audit fix --force`
Will install react-scripts@3.0.1, which is a breaking change
node_modules/svgo/node_modules/nth-check
  css-select  <=3.1.0
  Depends on vulnerable versions of nth-check
  node_modules/svgo/node_modules/css-select
    svgo  1.0.0 - 1.3.2
    Depends on vulnerable versions of css-select
    node_modules/svgo
      @svgr/plugin-svgo  <=5.5.0
      Depends on vulnerable versions of svgo
      node_modules/@svgr/plugin-svgo
        @svgr/webpack  4.0.0 - 5.5.0
        Depends on vulnerable versions of @svgr/plugin-svgo
        node_modules/@svgr/webpack
          react-scripts  >=2.1.4
          Depends on vulnerable versions of @svgr/webpack
          Depends on vulnerable versions of resolve-url-loader
          node_modules/react-scripts

postcss  <8.4.31
Severity: moderate
PostCSS line return parsing error - https://github.com/advisories/GHSA-7fh5-64p2-3v2j
```

Fonte: Autores

O *npm-audit* informou a vulnerabilidade implantada para viabiliza o XSS no Bootstrap; e outras vulnerabilidades na versão do *nth-check*, do *postcss* e do *ws*. No repositório de *Backend*, esta auditoria permitiu o relatório da Figura 40.

Figura 40 – Componentes vulneráveis identificados pelo *npm* no *Backend* da POC1

```
# npm audit report

braces  <3.0.3
Severity: high
Uncontrolled resource consumption in braces - https://github.com/advisories/GHSA-grv7-fg5c-xmjg
fix available via `npm audit fix`
node_modules/braces

express  <4.19.2
Severity: moderate
Express.js Open Redirect in malformed URLs - https://github.com/advisories/GHSA-rv95-896h-c2vc
fix available via `npm audit fix`
node_modules/express

2 vulnerabilities (1 moderate, 1 high)

To address all issues, run:
  npm audit fix
```

Fonte: Autores

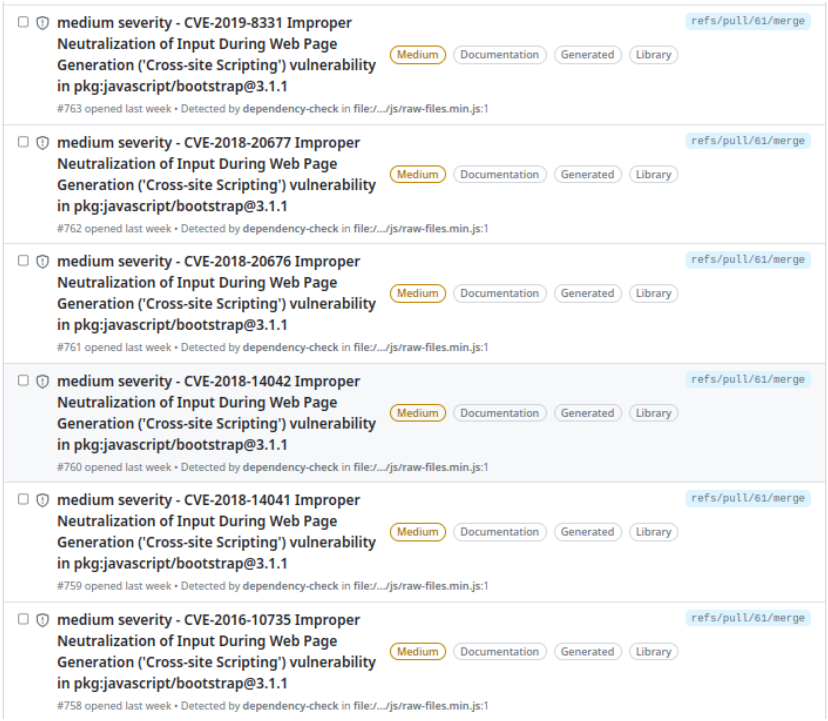
De acordo com o relatório, os pacotes *braces* e *express* estão sendo utilizados em versões com vulnerabilidades altas e moderadas, respectivamente.

### Checagem de Dependências

Owasp Dependency Check/Snyk

Na etapa de verificação de dependências vulneráveis, o Snyk, o Dependabot e o OWASP *Dependency Check* geraram relatórios. O relatório 41 contém as dependências vulneráveis encontradas pelo OWASP *Dependency Check* na aba *Security* do Github.

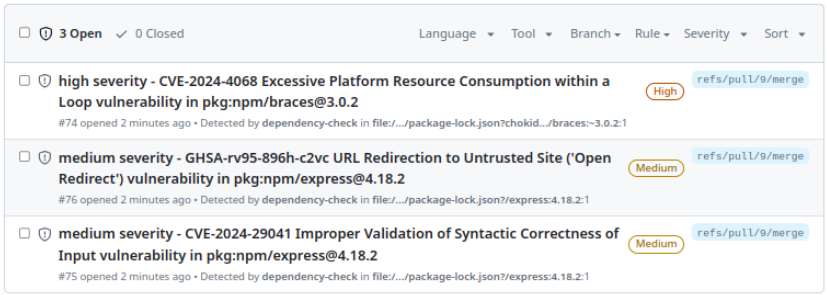
Figura 41 – Alertas do OWASP *Dependency Check* no *Frontend* da POC1




Fonte: Autores

Assim como o *npm-audit*, o OWASP *Dependency Check* também identificou no *Frontend* as dependências vulneráveis do *bootstrap*, do *nth-check* e do *postcss*. Entretanto, encontrou outras adicionais nos pacotes *css-what:3.4.2*, *customize.min.js*, *customizer.js*, *micromatch:4.0.7*, *ejs:3.1.10*, *jquery.js* e *raw-files.min.js*. Para o repositório de *Backend*, o OWAPS *Dependency Check* encontrou três dependências vulneráveis (vide Figura 42).

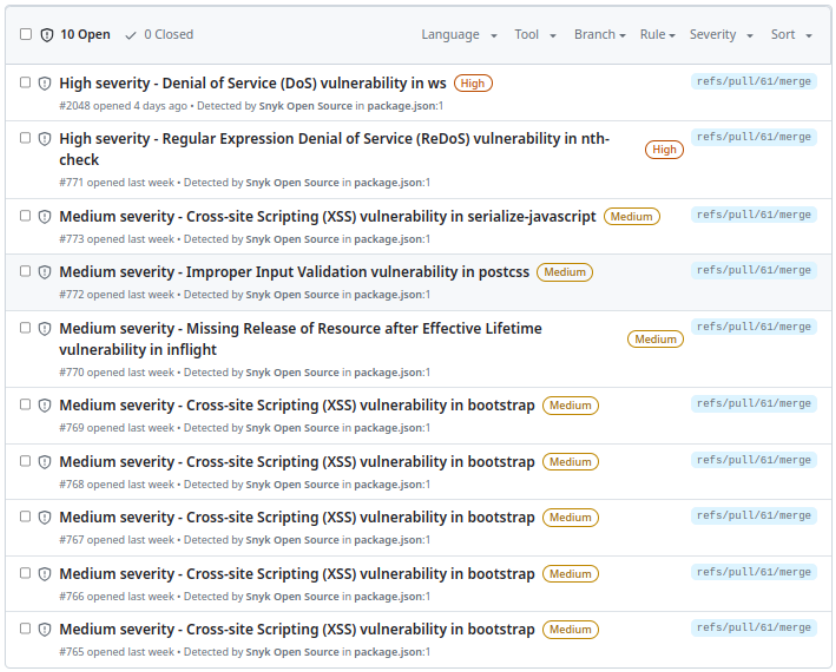
Figura 42 – Alertas do OWASP *Dependency Check* no *Backend* da POC1




Fonte: Autores

No *Backend*, o OWASP *Dependency Check* fez uma avaliação semelhante ao *npm-audit*. No entanto, apresentou uma vulnerabilidade a mais para o pacote *express*. O Snyk também gerou relatórios como na Figura 43.

Figura 43 – Relatório Snyk no *Frontend*




Fonte: Autores

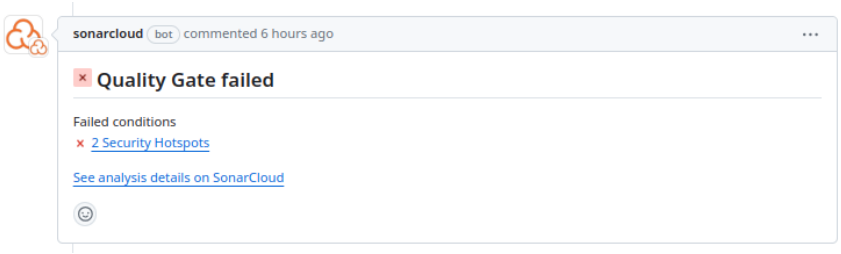
O Snyk percebeu as vulnerabilidades anteriormente vistas pelas outras ferramentas de análise de dependências, *bootstrap*, *ws*, *nth-check*, *serialize-javascript* e *postcss*, mas gerou um alerta informando a Falta de Liberação de recurso após Vulnerabilidade de Vida Útil Efetiva. No *Backend*, avisou sobre o *express* e a vulnerabilidade de redirecionamento aberto.

## SAST

### SonarCloud

No *step* de execução do SonarCloud no repositório de *Frontend*, o *bot* do Sonar apresentou o comentário da Figura 44 no *Pull Request* aberto.

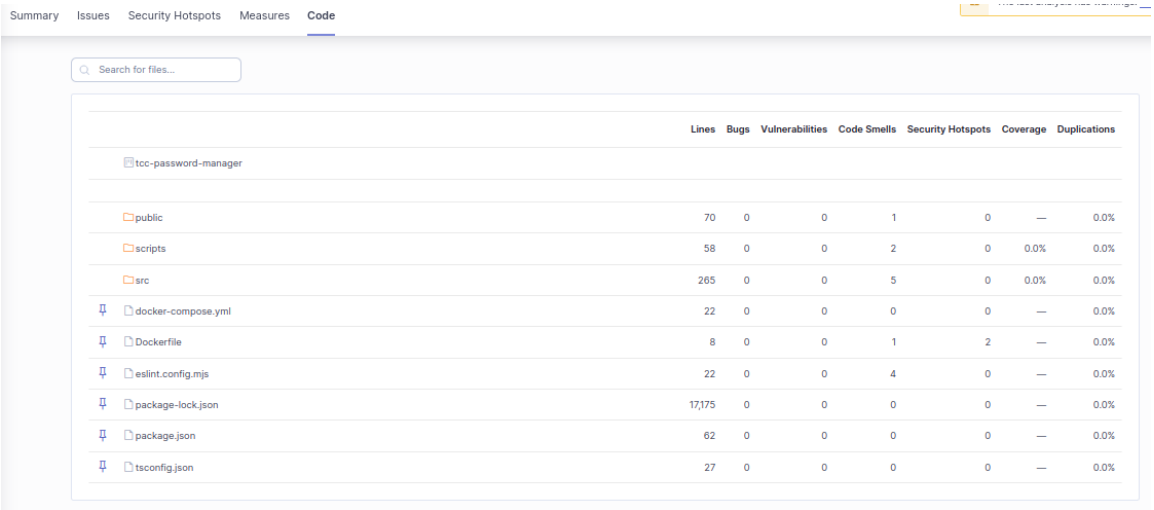
Figura 44 – Comentário do SonarCloud no PR do *Frontend* da POC1




Fonte: Autores

Foram encontrados dois *Security Hotspots*. Esse valor é maior do que o estabelecido para passar no Portão de Qualidade (0 *Security Hotspots*). Para a inspeção mais detalhada, o SonarCloud pode ser acessado, como demonstra a Figura 45.

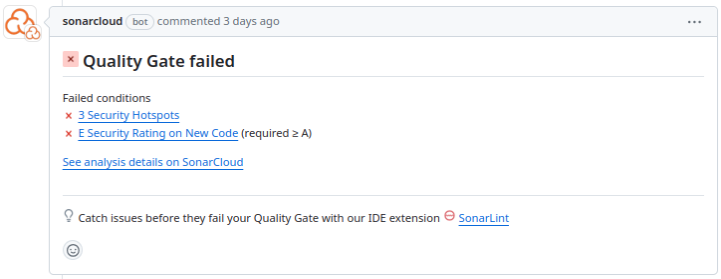
Figura 45 – Detalhes do SonarCloud no *Frontend* da POC1




Fonte: Autores

Os *Security Hotspots* encontrados pelo SonarCloud estão no arquivo *Dockerfile* e não há vulnerabilidades em outros caminhos. Esta análise realizada no *Frontend* também é feita no *Backend*, conforme a Figura 67.

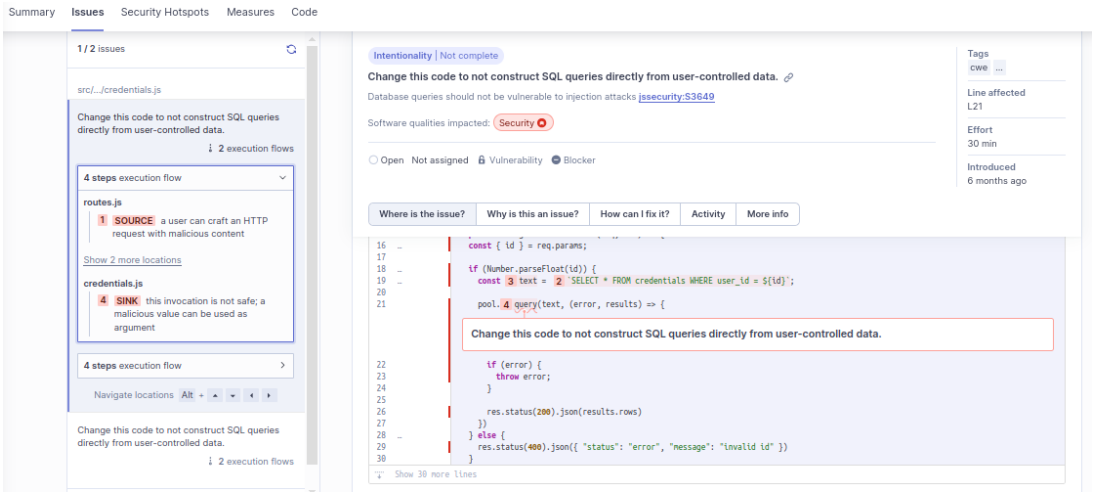
Figura 46 – Comentário do SonarCloud no PR do *Backend* da POC1




Fonte: Autores

No *Backend*, o Sonar identificou três *Security Hotspots*, resultando em falha no *Quality Gate*, que exige o valor 0. Outras falhas de segurança são encontradas no *Backend*, não como *Security Hotspots*, mas como vulnerabilidades, como especifica a Figura 47.

Figura 47 – Comentário do SonarCloud no PR do *Frontend* da POC1



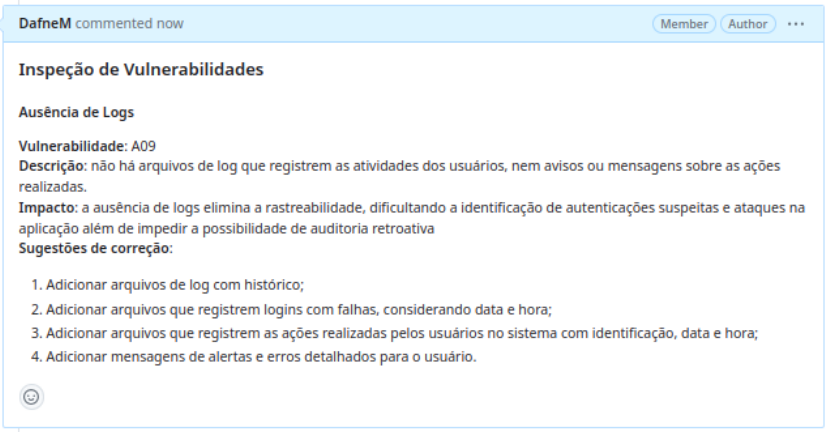

Fonte: Autores

O SonarCloud identificou o caminho e a linha de código das Injeções (*SQL Injection*) implementadas pelos autores descritas na Seção 5.4.1.1.

### Inspeção de Código/Testes de Sistema

Por fim, após toda a análise de relatórios das etapas de *Build*, SCA e SAST, foi feito o *pair review* com inspeção de código e testes de sistema. A Figura 48 apresenta um relatório de inspeção que identificou ausência de *logs* no sistema.

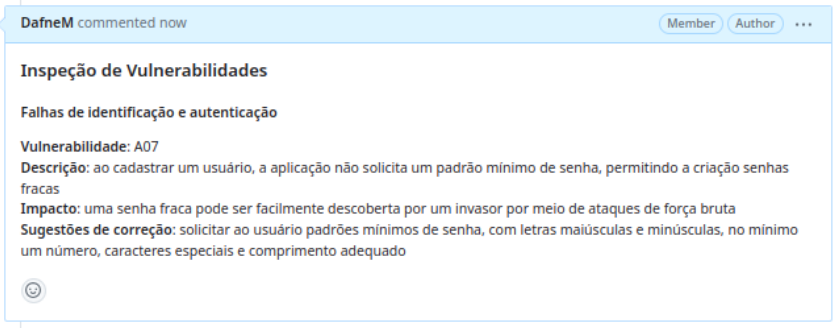
Figura 48 – Relatório de Inspeção de Identificação de Ausência de *Logs* na POC1




Fonte: Autores

O relatório de inspeção de ausência de *logs* foi relatado pelo revisor após análise do código fonte, com a vulnerabilidade, a descrição, o impacto e algumas sugestões de correção. Outras vulnerabilidades foram percebidas por meio de Testes de Sistema. O comentário feito pelo revisor na Figura 49 demonstra o resultado de um teste de sistema feito na ramificação pelo revisor.

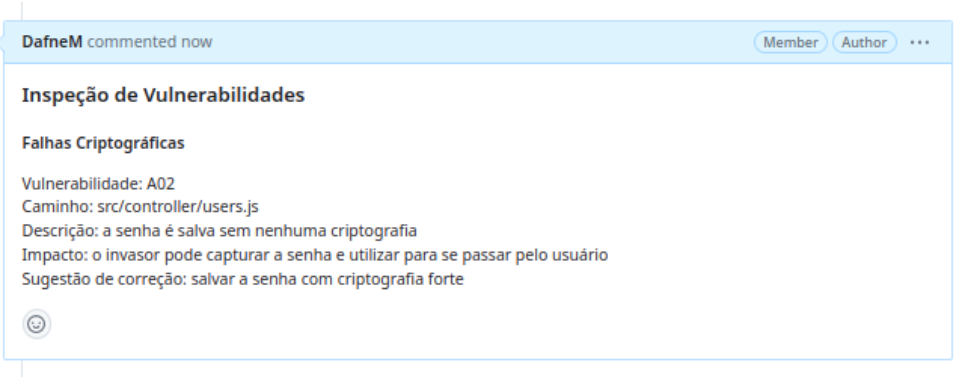
Figura 49 – Relatório de Inspeção de Identificação de Senha fraca na POC1




Fonte: Autores

Nos testes, o revisor capturou a possibilidade do usuário cadastrar senhas sem nenhum requisito mínimo exigido. A vulnerabilidade A02 foi codificada pelos autores e também identificada pelo revisor no relatório de Inspeção da Figura 50.

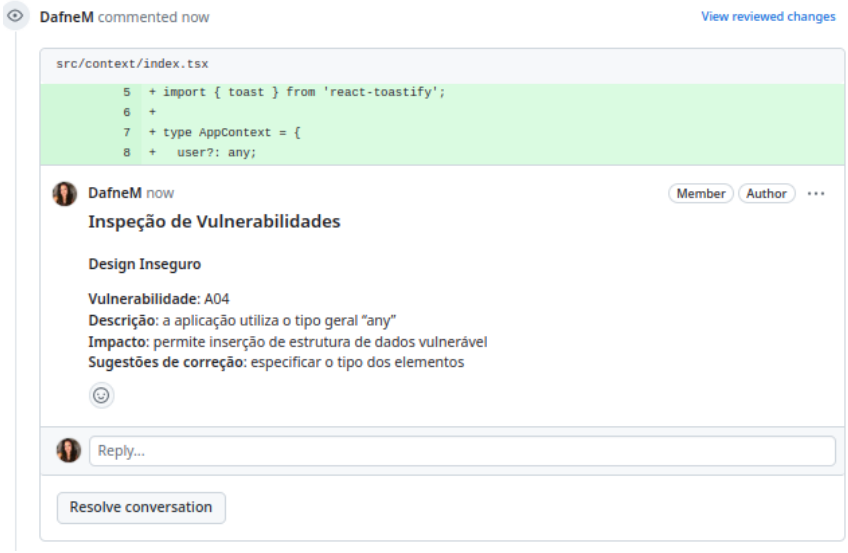
Figura 50 – Relatório de Inspeção de Identificação de Senha em texto claro na POC1




Fonte: Autores

Outra observação foi a utilização de tipos *any* na aplicação como na Figura 51.

Figura 51 – Tipo *any*




Fonte: Autores

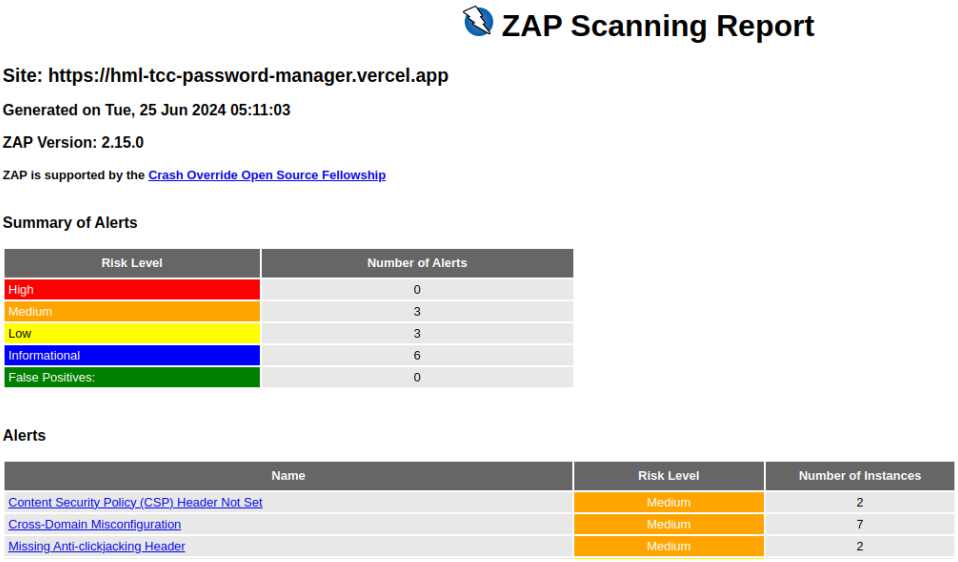
Para finalizar, o revisor acrescentou a anotação de que a ausência de definição de tipo pode resultar em estruturas vulneráveis. Seguido da revisão, o PR foi aceito, mesmo com os alertas de segurança, para que todas as verificações DAST fossem realizadas.

## DAST

### Owasp ZAP

O relatório de *scanner passivo* (OWASP ZAP *Baseline Scan*) descrito em seu formato HTML na Figura 52.

Figura 52 – Resultados do ZAP *Baseline Scan* na POC1



**ZAP Scanning Report**

**Site: https://hml-tcc-password-manager.vercel.app**

**Generated on Tue, 25 Jun 2024 05:11:03**

**ZAP Version: 2.15.0**

**ZAP is supported by the Crash Override Open Source Fellowship**

**Summary of Alerts**

| Risk Level | Number of Alerts |
|---|---|
| High | 0 |
| Medium | 3 |
| Low | 3 |
| Informational | 6 |
| False Positives: | 0 |

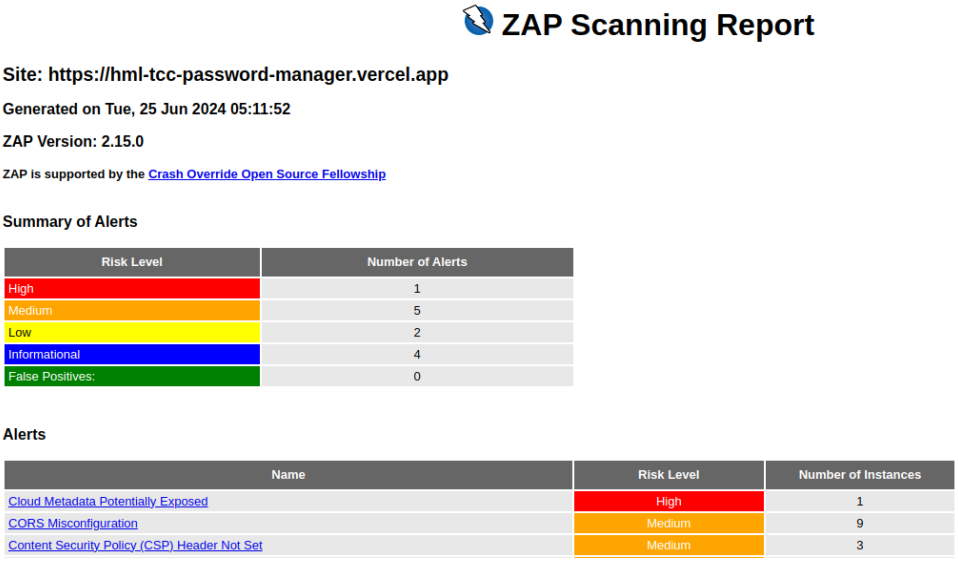
**Alerts**

| Name | Risk Level | Number of Instances |
|---|---|---|
| Content Security Policy (CSP) Header Not Set | Medium | 2 |
| Cross-Domain Misconfiguration | Medium | 7 |
| Missing Anti-clickjacking Header | Medium | 2 |

Fonte: Autores

A varredura do tipo *Baseline* identificou três alertas médios, três baixos e seis informacionais. Sendo uma delas a ausência de configuração de domínio cruzado, inserida pelos autores para representar a A05 da OWASP. A varredura do tipo *Full Scan ZAP* também gerou relatório do *scan ativo*. Detalhes podem ser vistos na Figura 53.

Figura 53 – Resultados do ZAP *Full Scan* na POC1



**ZAP Scanning Report**

**Site: https://hml-tcc-password-manager.vercel.app**

**Generated on Tue, 25 Jun 2024 05:11:52**

**ZAP Version: 2.15.0**

**ZAP is supported by the Crash Override Open Source Fellowship**

**Summary of Alerts**

| Risk Level | Number of Alerts |
|---|---|
| High | 1 |
| Medium | 5 |
| Low | 2 |
| Informational | 4 |
| False Positives: | 0 |

**Alerts**

| Name | Risk Level | Number of Instances |
|---|---|---|
| Cloud Metadata Potentially Exposed | High | 1 |
| CORS Misconfiguration | Medium | 9 |
| Content Security Policy (CSP) Header Not Set | Medium | 3 |

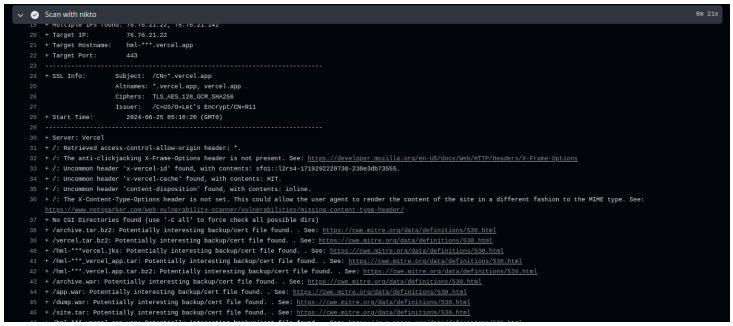
Fonte: Autores

O *Full Scan* identificou um alerta de criticidade alta, cinco de criticidade média, dois de criticidade baixa e quatro apenas informacionais, sendo identificada a ausência de configuração do CORS.

### Nikto

A *action* do Nikto falhou em sua execução, o que era esperado, visto que a configuração foi definida para falhar ao encontrar itens a serem reportados. O resultado foi percebidos na Figura 54.

Figura 54 – Execução da *action* do Nikto na POC1




Fonte: Autores

Com a análise dos relatórios DAST e a realização de novos Testes de Sistema, o PR foi mesclado para a produção. Neste primeiro momento, as vulnerabilidades foram propositalmente implantadas em produção. Os autores iniciaram então a resolução das vulnerabilidades.

**Resolução de vulnerabilidades**

Foi utilizada a ferramenta *OWASP Threat Dragon* para a modelagem de ameaças da aplicação. Foi utilizado o modelo STRIDE para a modelagem.

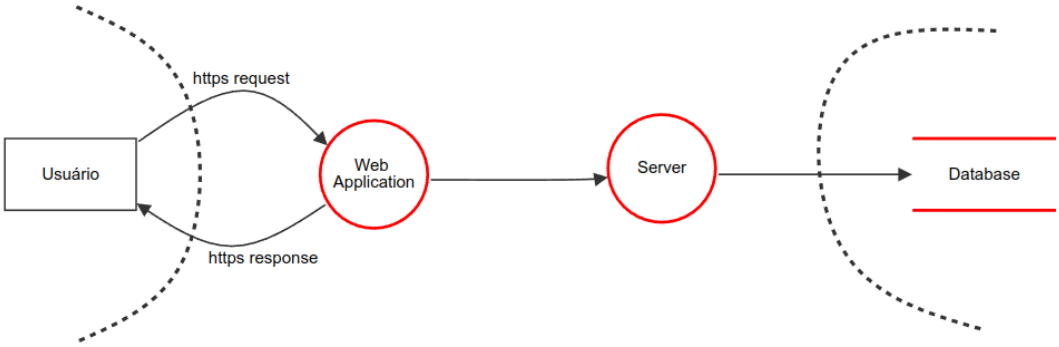
O STRIDE é um modelo para identificação de ameaças de segurança, servindo como um mnemônico para diferenciar seis tipos diferentes categorias de vulnerabilidades, sendo elas:

- *Spoofing*: Quando uma entidade assumem a identidade de uma outra entidade. Normalmente acontece por meio de uma forma de falsificação das credenciais de autenticação;
- *Tampering*: Alguma alteração não autorizada de dados da aplicação. Pode ocorrer durante transmissão de informações;
- *Repudiation*: Realização de ações na aplicação sem registro algum. Abre a possibilidade de que usuários realizem ações na aplicação que não possam ser identificadas;
- *Information Disclosure*: Exposição de dados não desejadas e autorizadas;
- *Denial of Service*: Interrupção dos serviços disponibilizados, e

- *Elevation of Privilege*: Quando um usuário consegue obter acesso a funcionalidades que não deveriam estar disponíveis para o perfil do usuário.

A Figura 55 realiza uma representação simplificada de um diagrama de modelagem de ameaças, muito usado para identificar possíveis vulnerabilidades e riscos em um sistema.

Figura 55 – Modelagem de Ameaças - Diagrama




Fonte: Autores

É possível também verificar, detalhadamente, as vulnerabilidades registradas. Cada vulnerabilidade foi distinguida por meio de um título, descrição, tipo, prioridade, *status* e formas de mitigação.

Na Figura 56, são apresentadas as vulnerabilidades relacionadas à camada da aplicação web.

Figura 56 – Modelagem de Ameaças - *Web*

Web Application (Process)

Camada de utilização do usuário

| Number | Title | Type | Priority | Status | Score | Description | Mitigations |
|---|---|---|---|---|---|---|---|
| 9 | Reconhecimento por senhas mais fracas | Information disclosure | High | Open | | Inicialmente a aplicação esconde as senhas dos usuarios, porém não existe uma padronização das senhas quando escondidas, assim é possível identificar credenciais com senhas com menos caracteres, facilitando a identificação por atacantes de credenciais potencialmente exploraveis a partir de uma ataque de força-bruta | Padronização visual das senhas dos usuários |
| 15 | Utilização de senhas fracas | Information disclosure | Medium | Open | | A aplicação permite a criação de contas sem uma padrão de segurança mínimo. | Utilização de requisitos para as senhas ao criar uma conta, como por exemplo:<br><br>- Mínimo de caracteres<br>- Utilização de simbolos/caracteres especiais<br>- Utilização de numeros<br>- Utilização de letras maiusculas e minusculas |
| 17 | Forja de solicitações do usuario (CSRF) | Tampering | Medium | Open | | Um ataque CSRF permite que um usuário realizar uma ação sem o consentimento por meio de algum link acessado por ele. Pode ser usado tanto para o recebimento das credenciais do usuario quando para a criação de novas credenciais | Utilização de tokens unicos por sessão (CSRF) |
| 22 | Controle de tentativas de login | Tampering | High | Open | | A aplicação não possui mecanismos para bloquear várias tentativas de login. Isso é um risco pois permite ataques de força-bruta, onde o usuário pode realizar várias tentativas sem nenhuma restrição. | Utilização de mecanismos visando evitar ataques de força-bruta, como por exemplo, utilização de CAPTCHAS |

Fonte: Autores

Na Figura 57, são apresentas as vulnerabilidades relacionadas à camada da aplicação servidor. Essa camada é diretamente relacionada com o banco de dados. As vulnerabilidades relacionadas ao banco de dados estão descritas na Figura 58.

Figura 57 – Modelagem de Ameaças - Servidor

**Server (Process)**

Recebe requisições e interage com o banco de dados

| Number | Title | Type | Priority | Status | Score | Description | Mitigations |
|---|---|---|---|---|---|---|---|
| 3 | Ausência de coleta de logs | Repudiation | Medium | Open | | A aplicação não realiza coleta de logs do usuário | Utilização de algum mecanismo de log para ações efetuadas pelo usuário |
| 5 | Acesso a dados de outros usuários | Spoofing | High | Open | | É possível acessar os dados de outros usuários a partir do ID dos mesmos | Utilização de tokens de autenticação para identificar o usuário e permitir somente o retorno dos seus próprios dados |
| 10 | Injeção | Spoofing | High | Open | | É possível realizar um ataque de injeção no endpoint de autenticação, permitindo assim operações inesperadas no banco de dados da aplicação. Um exemplo é o acesso a conta de um usuário sem conhecimento da senha do mesmo. | Validação e Sanitização dos dados enviados pelo usuário |
| 14 | CORS | Spoofing | Medium | Open | | Ausência de configuração de CORS permitindo que qualquer aplicação realize requisições | Configuração do CORS para definir os domínios que podem realizar requisições para a API |
| 21 | Configuração de Headers | Tampering | Medium | Open | | Configuração de headers basicos a serem usados na aplicação para prevenir diversas vulnerabilidades exploráveis. | Validar a utilização de headers. Exemplos:<br><br>- X-XSS-Protection<br>- Content-Security-Policy (CSP)<br>- X-Frame-Options<br>- Strict-Transport-Security |
| 23 | Limitação e controle de requisições | Denial of service | Medium | Open | | Não existe na aplicação mecanismos de controle e prevenção de envio de várias requisições, o que permite a realização de ataques de negação de serviço. Essa vulnerabilidade também está relacionada a ataques de força bruta, já que não já nenhum bloqueio na repetição de tentativas de login com os dados de um usuário. | Utilização de regras de firewall e de sistemas de prevenção (IPS - Intrusion Prevention Systems) |

Fonte: Autores

Figura 58 – Modelagem de Ameaças - Banco de Dados

**Database (Store)**

Armazena os dados dos usuarios

| Number | Title | Type | Priority | Status | Score | Description | Mitigations |
|---|---|---|---|---|---|---|---|
| 1 | Senhas de credenciais em plaintext | Information disclosure | High | Open | | A aplicação armazena as senhas das credenciais cadastradas pelos usuários em plaintext, possibilitando que uma invasão exponha os dados sensíveis dos usuários sem nenhuma camada extra de proteção. | Criptografia das senhas das credenciais dos usuários. |
| 6 | Utilização de ID sequencial | Information disclosure | Medium | Open | | A aplicação utiliza IDs numericos sequenciais (1,2,3,4...) para registrar novas contas. Isso facilita que atacantes possam utilizar do ID do usuario para realização de ataques. | Alterar para IDs randomicos, como por exemplo, UUIDs. |

Fonte: Autores

Com a implantação das vulnerabilidades em produção, os desenvolvedores corrigiram estas vulnerabilidades seguindo a ordem estabelecida:

ORDEM DE CORREÇÃO

- A03:2021 - Injeção - 7.15
- A04:2021 - *Design* Inseguro - 6.78
- A07:2021 – Falhas de Identificação e Autenticação - 6.50
- A01:2021 – Quebra de Controle de Acesso - 5.93
- A09:2021 – Falhas de Registro e Monitoramento de Segurança - 4.99
- A06:2021 - Componentes Vulneráveis - 2.2

Gerando as seguintes correções:

- *Commit* 490c2deb9ace6ae7088679fd1d5cf07afca6705d[1]: (A03 - Injeção) Sanitização dos valores dos parâmetros.
- *Commit* a4678f4ed7e94ae58a249be7642e6f51433377d3[2]: (A03 - Injeção) Sanitização dos valores dos parâmetros.
- *Commit* 7e7ef6a187a9b8d4979f8e9b1e46a07451f3fd7e[3]: (A04 - *Design* Inseguro) Utilização de tipagem no código do *frontend*.
- *Commit* 761f6e7d2e1ac2d8344541782faf112e23566227[4]: (A04 - Falhas de identificação e autenticação) Foram definidos, na aplicação do Frontend, padrões mínimos de senha. Para tanto, foi utilizada a biblioteca zxcvbn, que consegue definir um *score* para uma senha. Com essa alteração, passou a ser exigido um *score* mínimo para a senha do usuário.
- *Commit* beeeddd6db8a3054f854de47313d408f5fb9049b[5]: (A01 - Quebra de Controle de Acesso) Envio do *token* do usuário, substituindo e trocando assim a passagem do parâmetro que originalmente era o identificador da conta do usuário.
- *Commit* 0cd2b3b457862d41930b95551fe1ef76a7655c01[6]: (A01 - Quebra de Controle de Acesso) Leitura do *token* de usuário para identificar o identificador do usuário.
- *Commit* 8d113f0724fa1d360cda0130757790e5fd142832[7]: (A01 - Quebra de Controle de Acesso) Adiciona *rate limit* nas chamadas da API.
- *Commit* e3ecc4fd5d4ff72055abd47c43928af2c8a6c57a[8]: (A09 - Falhas de registro e monitoramento de segurança) Com auxílio das ferramentas, foi implementado um sistema de *logs*, armazenando histórico de chamadas de *endpoints*, assim como dados do solicitante, além de textos uteis para realizar *debug* de eventuais problemas. Todas as informações são salvas em uma pasta com nome "*logs*"na raiz do projeto.

[1] Disponível em: <https://github.com/tcc-lucas-dafne/tcc-password-manager-api/pull/11/commits/490c2deb9ace6ae7088679fd1d5cf07afca6705d> e Último acesso em: Julho 2024.
[2] Disponível em: <https://github.com/tcc-lucas-dafne/tcc-password-manager-api/pull/11/commits/a4678f4ed7e94ae58a249be7642e6f51433377d3> e Último acesso em: Julho 2024.
[3] Disponível em: <https://github.com/tcc-lucas-dafne/tcc-password-manager/pull/68/commits/7e7ef6a187a9b8d4979f8e9b1e46a07451f3fd7e> e Último acesso em: Julho 2024.
[4] Disponível em: <https://github.com/tcc-lucas-dafne/tcc-password-manager/pull/67/commits/761f6e7d2e1ac2d8344541782faf112e23566227> e Último acesso em: Julho 2024
[5] Disponível em: <https://github.com/tcc-lucas-dafne/tcc-password-manager/pull/69/commits/beeeddd6db8a3054f854de47313d408f5fb9049b> e Último acesso em: Julho 2024
[6] Disponível em: <https://github.com/tcc-lucas-dafne/tcc-password-manager-api/pull/12/commits/0cd2b3b457862d41930b95551fe1ef76a7655c01> e Último acesso em: Julho 2024
[7] Disponível em: <https://github.com/tcc-lucas-dafne/tcc-password-manager-api/pull/12/commits/8d113f0724fa1d360cda0130757790e5fd142832> e Último acesso em: Julho 2024
[8] Disponível em: <https://github.com/tcc-lucas-dafne/tcc-password-manager-api/pull/13/commits/e3ecc4fd5d4ff72055abd47c43928af2c8a6c57a> e Último acesso em: Julho 2024

- *Commit* f9c3cae90480c38b9fb7a554fdd0ba6c402cfc58[9]: (A06 - Componentes Vulneráveis) Atualização de dependências no *frontend*. Foi possível realizar algumas atualizações, e correções, que foram detectadas pelo *npm*. Porém, não foi possível a correção de algumas versões. Isso aconteceu, pois a atualização de algumas versões potencialmente quebrariam outras dependências.

- *Commit* f9c3cae90480c38b9fb7a554fdd0ba6c402cfc58[10]: (A06 - Componentes Vulneráveis) Para a parte do *backend*, a atualização foi completa. Todas as dependências que estavam desatualizadas foram atualizadas.

#### 5.4.1.3 Análise de resultados

O detalhamento sobre a análise de resultados da Solução abordada anteriormente considerou a abordagem híbrida: parte quantitativa, e parte qualitativa. Nesse caso, o processo ocorreu orientando-se pelo Método de Análise de Resultados, descrito no Capítulo 4 - Metodologia.

Para avaliar a eficácia do *pipeline* DevSecOps, as vulnerabilidades implantadas propositalmente pelos autores foram categorizadas quanto à sua detecção pelo *pipeline*. No Quadro 4, são apresentadas todas essas vulnerabilidades, permitindo uma análise pormenorizada da eficiência das práticas de segurança introduzidas.

Quadro 4 – Vulnerabilidades encontradas pelo *pipeline*

| OWASP | ID | Vulnerabilidade | Identificada |
|---|---|---|---|
| A01 | P1-001-A01 | Acesso às credenciais de outro usuário | |
| A01 | P1-002-A01 | Token sem expiração | |
| A01 | P1-003-A01 | Ausência de middleware de autenticação | |
| A01 | P1-004-A01 | Não existe nenhum *rate limit* nas chamadas da API | X |
| A02 | P1-005-A02 | Senhas são armazenadas como *plain text* | X |
| A02 | P1-006-A02 | Ausência de validação da assinatura do token | |
| A03 | P1-007-A03 | *SQL Injection* | X |
| A04 | P1-008-A04 | A aplicação usa o *type any* | X |
| A05 | P1-009-A05 | Ausência de configuração de CORS | X |
| A06 | P1-010-A06 | Componentes vulneráveis (bootstrap) | X |
| A07 | P1-011-A04 | Ausência de requisitos mínimos de senha | X |
| A09 | P1-012-A09 | Ausência de Logs | X |

Fonte: Autoria própria.

[9] Disponível em: <https://github.com/tcc-lucas-dafne/tcc-password-manager/pull/70/commits/f9c3cae90480c38b9fb7a554fdd0ba6c402cfc58> e Último acesso em: Julho 2024

[10] Disponível em: <https://github.com/tcc-lucas-dafne/tcc-password-manager-api/pull/13/commits/e3ecc4fd5d4ff72055abd47c43928af2c8a6c57a> e Último acesso em: Julho 2024

Estas vulnerabilidades foram reconhecidas em várias etapas do *pipeline* DevSecOps, como especificado na Seção Aplicação do *Pipeline*. O Quadro 5 agrupa em etapas as detecções realizadas.

Quadro 5 – Vulnerabilidades encontradas pelo *pipeline*

| Vulnerabilidade | Auditoria | Dependências | SAST | Inspeção/ Testes Sistema | DAST |
|---|---|---|---|---|---|
| P1-001-A01 | | | | | |
| P1-002-A01 | | | | | |
| P1-003-A01 | | | | | |
| P1-004-A01 | | | X | | |
| P1-005-A02 | | | | X | |
| P1-006-A02 | | | | | |
| P1-007-A03 | | | X | | |
| P1-008-A04 | | | | X | |
| P1-009-A05 | | | | | X |
| P1-010-A06 | X | X | | X | |
| P1-011-A07 | | | | X | |
| P1-012-A09 | | | | X | |

Fonte: Autoria própria.

Dentro das etapas, as vulnerabilidades foram localizadas por diferentes métodos de detecção. O Quadro 6 discorre sobre o método de identificação de cada vulnerabilidade implantada. As vulnerabilidades, cujo método estiver preenchido com "-" não foram encontradas pelo *pipeline* por método algum.

Quadro 6 – Vulnerabilidades encontradas pelo *pipeline*

| OWASP | Vulnerabilidade | Método de identificação |
|---|---|---|
| A01 | P1-001-A01 | - |
| A01 | P1-002-A01 | - |
| A01 | P1-003-A01 | - |
| A01 | P1-004-A01 | CodeQL |
| A02 | P1-005-A02 | Inspeção |
| A02 | P1-006-A02 | - |
| A03 | P1-007-A03 | SonarCloud |
| A04 | P1-008-A04 | Inspeção |
| A05 | P1-009-A05 | ZAP |
| A06 | P1-010-A06 | npm-audit/OWASP Dependency Check/Snyk |
| A07 | P1-011-A07 | Inspeção/Testes de sistema |
| A09 | P1-012-A09 | Inspeção |

Fonte: Autoria própria.

Apenas uma das vulnerabilidades (A01) foi identificada pelo CodeQL. O OWASP ZAP detectou a vulnerabilidade A05, enquanto a A06 foi descoberta por meio de vali-

dação de dependências, e a A03 pelo Sonar. As vulnerabilidades A02, A04 e A09 foram encontradas durante a inspeção, e a A07 foi detectada nos testes de sistema.

Ao somar, foram introduzidas no total 12 vulnerabilidades na Prova de Conceito 1. A Tabela 2 resume as quantidades de vulnerabilidades implantadas e encontradas pelo *pipeline*, implantadas e não verificadas e total.

Tabela 2 – Quantidade de vulnerabilidades encontradas pelo *pipeline*

| **Verificadas** | **Não verificadas** | **Total** |
|---|---|---|
| 8 | 4 | 12 |

Fonte: Autoria própria.

A Equação 4 apresenta a taxa de vulnerabilidades verificadas pelo *pipeline* desenvolvido.

$$\text{taxa de vulnerabilidades verificadas} = \frac{8}{12} \tag{4}$$

$$\text{taxa de vulnerabilidades verificadas} \approx 0{,}67$$

À medida que o valor desta métrica se aproxima de 1, indica que a maioria das vulnerabilidades foi verificada, evidenciando a eficácia do *pipeline* nesta Prova de Conceito 1.

Além das vulnerabilidades descritas na Seção 5.4.1.1, o *pipeline* também revelou adicionais. O Quadro 3 resume a quantidade de vulnerabilidades extras encontrada em cada repositório da aplicação.

Tabela 3 – Vulnerabilidades extras

| **Repositório** | **Total Econtradas** | **Verificadas** | **Extras** |
|---|---|---|---|
| Frontend | 181 | 5 | 176 |
| Backend | 6 | 3 | 3 |
| Total | 187 | 8 | 179 |

Fonte: Autoria própria.

A maior parte das vulnerabilidades encontradas no repositório de *Frontend* foi de itens do Nikto, sendo 152 itens reportados por ele, a maioria de arquivos de certificado e chaves públicas e privadas. As outras 24 vulnerabilidades foram encontradas nas dependências vulneráveis, nas *secrets* e nos testes ZAP. As vulnerabilidades extras do *Backend* são de dependências vulneráveis. A Equação 5 apresenta a quantidade total de vulnerabilidades percebidas pelo *pipeline*.

$$\text{total de vulnerabilidades percebidas pelo } \textit{pipeline} \approx 187 \tag{5}$$

A Equação 6 apresenta a quantidade de vulnerabilidades extras.

$$\text{vulnerabilidades extras} \approx 179 \tag{6}$$

Com base na taxa de vulnerabilidades verificadas e na quantidade de vulnerabilidades extras identificadas, o *pipeline* demonstrou eficácia em alcançar seu objetivo na Prova de Conceito 1.

### 5.4.2 POC 2 - Aplicação Bancária

Guiando-se pelo Método Orientado a Provas de Conceito, estabelecido no Capítulo 4 - Metodologia, segue o detalhamento inerente à segunda prova de conceito. As mesmas ponderações realizadas para a primeira prova de conceito são consideradas aqui.

#### 5.4.2.1 Definição da PoC

Organizada em Requisitos da PoC, Sistema Objeto de Estudo e Vulnerabilidades OWASP Top 10.

**Requisitos da PoC**

Os requisitos funcionais definidos para a aplicação vulnerável desta PoC são:

- O usuário deve conseguir se cadastrar para acessar a aplicação;
- O usuário deve realizar acesso com *login* utilizando senha;
- O usuário deve conseguir acessar o saldo da sua conta;
- O usuário deve conseguir transferir dinheiro de uma conta para outra, e
- O usuário deve poder solicitar aumento de limite de cartão.

**Sistema Objeto de Estudo - *Internet Banking***

A escolha por esse tipo de aplicação deu-se devido à segurança envolvida nela. Até hoje, alguns bancos obrigam os usuários a instalarem módulos de segurança em seus navegadores para terem acesso às suas contas. Isso é uma tentativa dos bancos de minimizarem ataques no computador do próprio cliente.

Uma aplicação bancária vulnerável é fatal tanto para o usuário, que pode ter seu dinheiro roubado por um invasor; quanto para a instituição financeira, que poderá ter de arcar com a perda do cliente, além da perda de credibilidade.

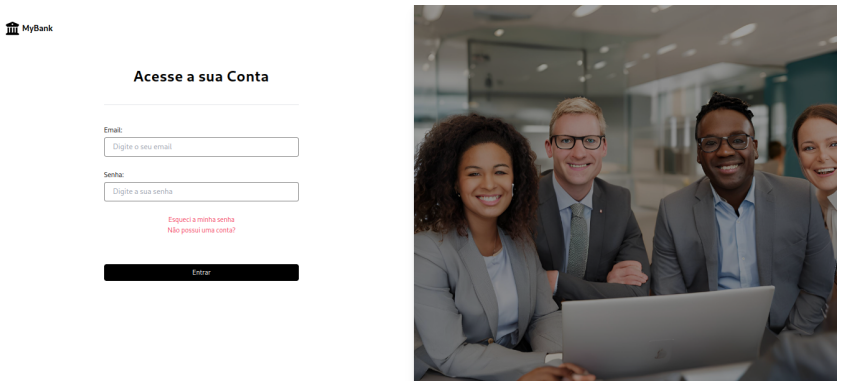
Diversas funcionalidades são fundamentais em uma aplicação bancária. A aplicação bancária proposta deve permitir o *login* do usuário e a visualização do perfil com seus dados pessoais. Além disso, o usuário deve conseguir também verificar o extrato da sua conta, realizar transferências e acessar o histórico de compras realizadas. Outra funcionalidade envolvida é o módulo relacionado ao cartão, que deve permitir a criação de um cartão virtual; a visualização do limite máximo disponível, e a opção de solicitar aumento neste limite.

A aplicação contendo vulnerabilidades pode ser acessada pelos repositórios, na ramificação *results*:

- Aplicação Bancária: https://github.com/tcc-lucas-dafne/tcc-bank
- Aplicação Bancária API: https://github.com/tcc-lucas-dafne/tcc-bank-api/

Ao acessar a aplicação, primeiro o usuário precisa fazer o *login* no sistema. A Figura 25 apresenta a tela de *Login* da aplicação.

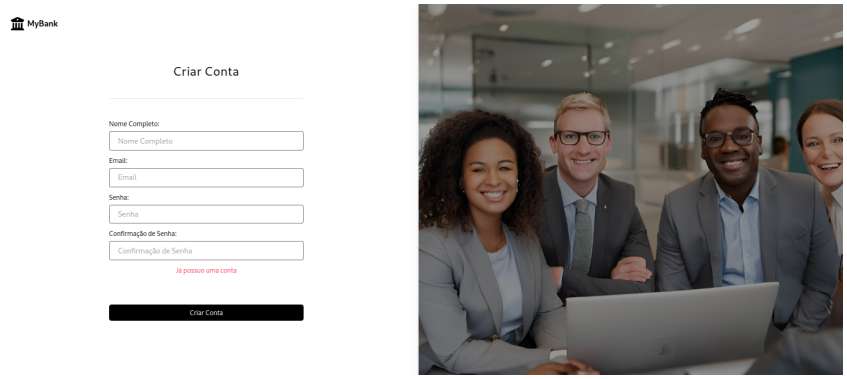
Figura 59 – Página de *Login* da Aplicação Bancária




Fonte: Autores

Caso este usuário não possua dados cadastrados no sistema, deve fazer o cadastro informando o nome completo, o *email*, a senha e uma confirmação desta senha. A Figura 60 apresenta a tela de Cadastro na aplicação.

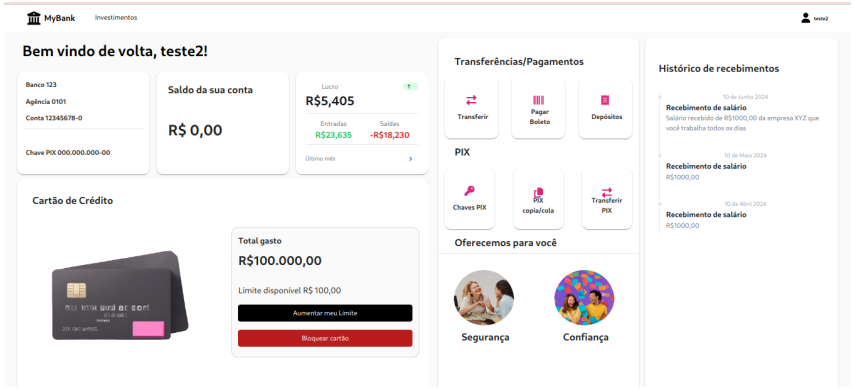
Figura 60 – Página de Cadastro da Aplicação Bancária




Fonte: Autores

Ao realizar o cadastro e o *login* na aplicação bancária, o usuário possui uma tela de visualização de *dashboard* com alguns dados fictícios para aproximá-lo da realidade dos sistemas financeiros. A Figura 61 mostra a *dashboard* da aplicação bancária.

Figura 61 – Página principal da Aplicação Bancária




Fonte: Autores

A partir desta *dashboard*, o usuário pode solicitar o aumento do limite do cartão; visualizar seu saldo; acessar os investimentos e seus dados cadastrados ao clicar no ícone do seu perfil na *Navbar*. Este sistema de bancário promete segurança e confiança ao usuário.

#### 5.4.2.1.1 Vulnerabilidades OWASP Top10 em Estudo

As vulnerabilidades Top 10 OWASP implantadas nesta aplicação foram A01, A02, A04, A03, A04, A05, A07, A08 A09 e A10.

### A01 Quebra de Controle de Acesso

São 2 vulnerabilidades na A01, o *token* sem expiração e a ausência de mecanismos de limitação nas chamadas API. Um exemplo de vetor é o *endpoint* de *login*, que pode ser chamado sem limite.

### A02 Falhas Criptográficas:

Uma outra é referente à senha cadastrada pelo usuário, que foi criptografada com a função de dispersão criptografia SHA1. Tal algoritmo é vulnerável, sendo facilmente decodificado, e não deve ser usado para armazenamento de senhas

### A03 Injeção:

Foram implantadas duas vulnerabilidades de injeção de código: os ataques XSS (*Cross-Site Scripting*) *Injection*.

O ataque de SQL *Injection* é encontrado no *Login* do Usuário, a partir do caminho: *src/controller/users.js*.

Código 13 – SQL *Injection*

```
1   const text = `
2     SELECT account.name, account.email, account.role, account_detail.*
3     FROM account
4     INNER JOIN account_detail ON account.account_id = account_detail.account_id
5     WHERE email='${email}' AND password='${hashedPassword}'
6   `;
```

Fonte: Autoria própria.

### A04 Design Inseguro:

A vulnerabilidade ocorrerá durante a manipulação dos dados do próprio usuário, possibilitando que ele visualize e manipule os dados livremente. Um exemplo disso é a solicitação de aumento de limite no cartão pelo usuário. Ao solicitar o aumento de limite, automaticamente o cartão passa a ter o limite solicitado sem qualquer verificação e aprovação por um perfil com acesso de administrador.

### A05 Configuração Incorreta de Segurança:

São duas: não há *headers* de segurança na aplicação e não há configuração do CORS.

### A07 Falhas de identificação e autenticação:

São quatro vulnerabilidades de Falhas de Identificação e Autenticação: *bank drops*, uma vez que a aplicação não possui nenhum mecanismo para identificar unicamente um usuário e também não possui validação dos dados passados, permitindo assim que um usuário possa se passar por outro e a criação de um perfil falso, deixando com que criminosos utilizem estas contas se passando por outras pessoas, até mesmo recebendo dinheiro resultante de transações ilícitas.

Não há múltiplos fatores de autenticação.

A aplicação não possui um padrão mínimo de senha. Assim o usuário pode utilizar uma senha fraca.

O Token JWT da sessão não é invalidado corretamente.

### A08 Falhas de Software e Integridade de Dados:

O *token* da Vercel é exposto no arquivo de CI/CD do *Github Actions*. Esta falha de integridade dos dados permite acesso indevido por usuários não autorizados a recursos privados.

### A09 Falhas de registro e monitoramento de segurança:

Não há *logs* na aplicação que ofereça a rastreabilidade das ações.

### A10 Falsificação de Solicitação do Lado do Servidor (SSRF):

Na página de perfil de usuário, na atualização de dados, é possível fazer o *upload* de imagens do perfil do usuário a partir de uma URL que não é verificada. Tal URL é executada no lado do servidor. No código, esta vulnerabilidade pode ser encontrada no caminho: *src/controller/users.js*.

Outra vulnerabilidade é o LFI (*Local File Inclusion*).

#### 5.4.2.2 Solução

Organizada em Aplicação do Pipeline, OWASP Dependency Check, OWASP Zap, Resolução de Vulnerabilidades e Análise de Resultados.

**Aplicação do *Pipeline***

Para a execução da Prova de Conceito 2 utilizando o *Pipeline* DevSecOps desenvolvido, o nome da ramificação no *Github* para a implementação das vulnerabilidades mencionadas na Seção 5.4.1.1 também foi *results* no *Frontend* e no *Backend*. Os arquivos de fluxo de trabalho (*workflows*) foram configurados semelhantes à POC1. Por último, houve a abertura do *Pull Request* (PR) para a ramificação *develop* para conduzir as análises estáticas.

**Auditoria**

npm-audit

Durante o Build/Auditoria, o *npm-audit* produziu o relatório da Figura 62 no repositório *Frontend*.

Figura 62 – Componentes vulneráveis identificados pelo *npm* no *Frontend* da POC2

```
# npm audit report

nth-check  <2.0.1
Severity: high
Inefficient Regular Expression Complexity in nth-check - https://github.com/advisories/GHSA-rp65-9cf3-cjxr
fix available via `npm audit fix --force`
Will install react-scripts@3.0.1, which is a breaking change
node_modules/svgo/node_modules/nth-check
  css-select  <=3.1.0
  Depends on vulnerable versions of nth-check
  node_modules/svgo/node_modules/css-select
    svgo  1.0.0 - 1.3.2
    Depends on vulnerable versions of css-select
    node_modules/svgo
      @svgr/plugin-svgo  <=5.5.0
      Depends on vulnerable versions of svgo
      node_modules/@svgr/plugin-svgo
        @svgr/webpack  4.0.0 - 5.5.0
        Depends on vulnerable versions of @svgr/plugin-svgo
        node_modules/@svgr/webpack
          react-scripts  >=2.1.4
          Depends on vulnerable versions of @svgr/webpack
          Depends on vulnerable versions of resolve-url-loader
          node_modules/react-scripts

postcss  <8.4.31
Severity: moderate
PostCSS line return parsing error - https://github.com/advisories/GHSA-7fh5-64p2-3v2j
fix available via `npm audit fix --force`
Will install react-scripts@3.0.1, which is a breaking change
node_modules/resolve-url-loader/node_modules/postcss
  resolve-url-loader  0.0.1-experiment-postcss || 3.0.0-alpha.1 - 4.0.0
  Depends on vulnerable versions of postcss
  node_modules/resolve-url-loader

8 vulnerabilities (2 moderate, 6 high)

To address all issues (including breaking changes), run:
  npm audit fix --force
```

Fonte: Autores

As vulnerabilidades encontradas na POC2 também foram encontradas nas dependências *nth-check* e *postcss* da POC1. No *Backend*, o *npm-audit* apresentou o relatório da Figura 63.

Figura 63 – Componentes vulneráveis identificados pelo *npm* no *Backend* da POC2


Fonte: Autores

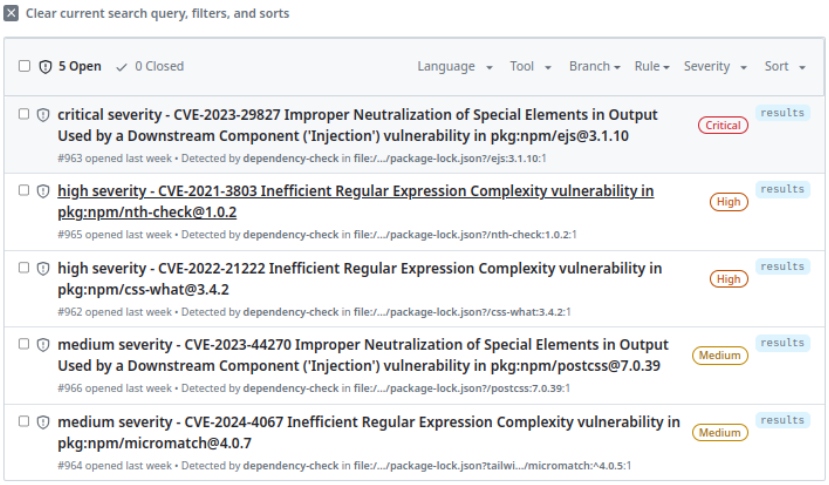
A ferramenta *npm-audit* não encontrou vulnerabilidades no *Backend* da aplicação.

### Checagem de Dependências

#### OWASP *Dependency Check*/Snyk

Ao longo da verificação de dependências vulneráveis, o relatório apresentado na Figura 64 exibe as vulnerabilidades identificadas pelo OWASP *Dependency Check* no *Frontend* da POC2, descritos na seção de segurança do GitHub.

Figura 64 – Alertas do OWASP *Dependency Check* no *Frontend* da POC2

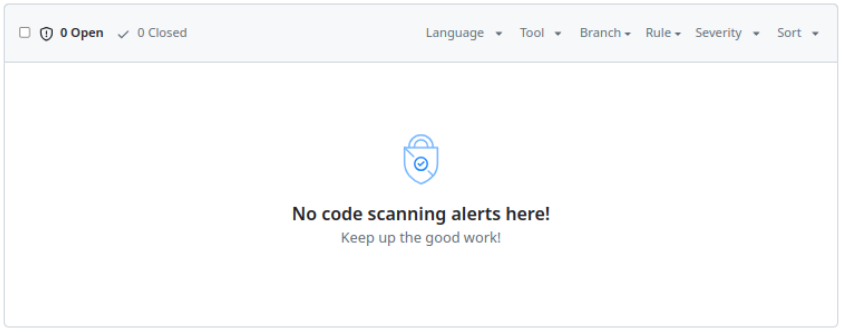



Fonte: Autores

O OWASP *Dependency Check* encontrou mais vulnerabilidades do que o *npm-audit*, explorando cinco vulnerabilidades, a vulnerabilidade crítica no pacote *ejs v3.1.10*, alta nos pacotes *nth-check v1.0.2*, *css-what v3.4.2*, e médias no *postcss v7.0.39* e no *micromatch v4.0.7*. O Snyk apresentou vulnerabilidade média no *serialize-javascript* e vulnerabilidade que pode levar ao esgotamento de recursos.

No *Backend*, o OWASP *Dependency Check* e o Snyk não encontraram dependências vulneráveis (vide Figura 65).

Figura 65 – Alertas do OWASP *Dependency Check* no *Backend* da POC2



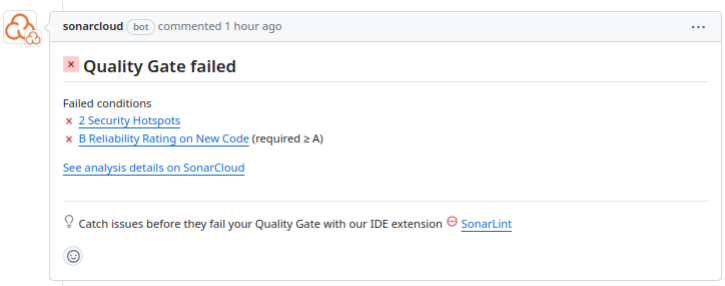

Fonte: Autores

## SAST

### SonarCloud

No decorrer dos testes SAST do repositório *Frontend*, o SonarCloud elaborou o relatório da Figura 66.

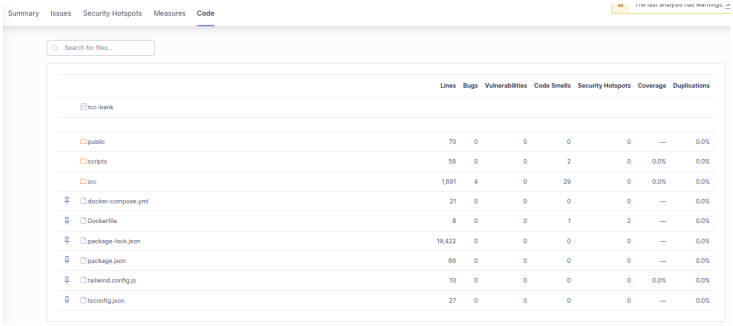
Figura 66 – Comentário do SonarCloud no PR do *Frontend* DA POC2




Fonte: Autores

A etapa do *Quality Gate* do SonarCloud no *Frontend* falhou, já que foram encontrados dois *Security Hotspots*. Mais detalhes são explicitados no SonarCloud (vide Figura 67).

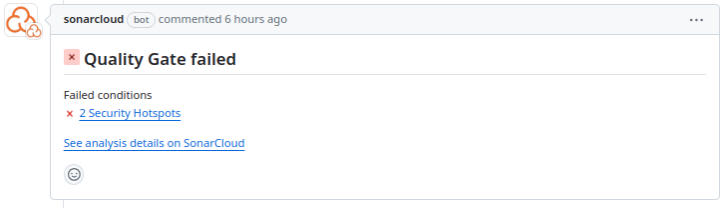
Figura 67 – Detalhes do SonarCloud no *Frontend* da POC2




Fonte: Autores

De modo semelhante à POC1, o *Dockerfile* é o arquivo que contém os *security hotspots* e não há vulnerabilidades em outros arquivos. Assim como no *Frontend*, no *Backend*, o código não passou pelo Portão de Qualidade, conforme a Figura 68.

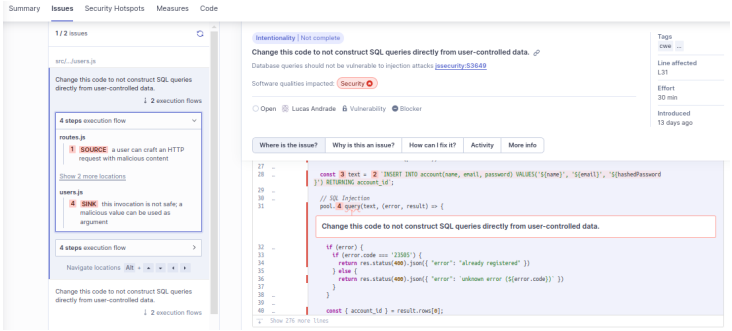
Figura 68 – Comentário do SonarCloud no PR do *Backend* da POC2




Fonte: Autores

Foram identificados pelo SonarCloud dois *Security Hotspots*, e os detalhes destes *Hotspots* e das vulnerabilidades podem ser vistos como na Figura 69.

Figura 69 – Comentário do SonarCloud no PR do *Backend* da POC2



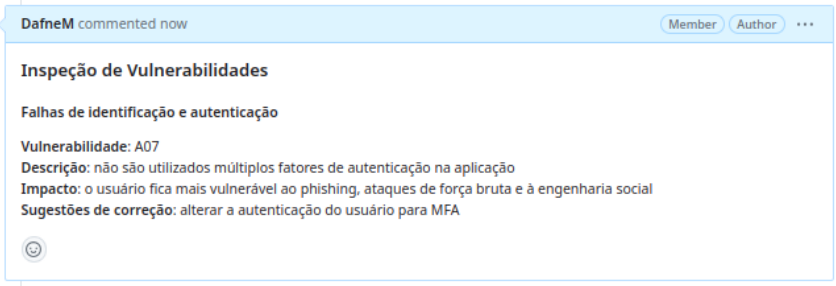

Fonte: Autores

O SonarCloud capturou as falhas de Injeção de Código SQL codificadas pelos autores para demonstrar a A03.

### Inspeção/Testes de Sistema

Na etapa de inspeção, o revisor adiciona comentários como o da Figura 70 para indicar o relatório de identificação de vulnerabilidade de ausência de MFA.

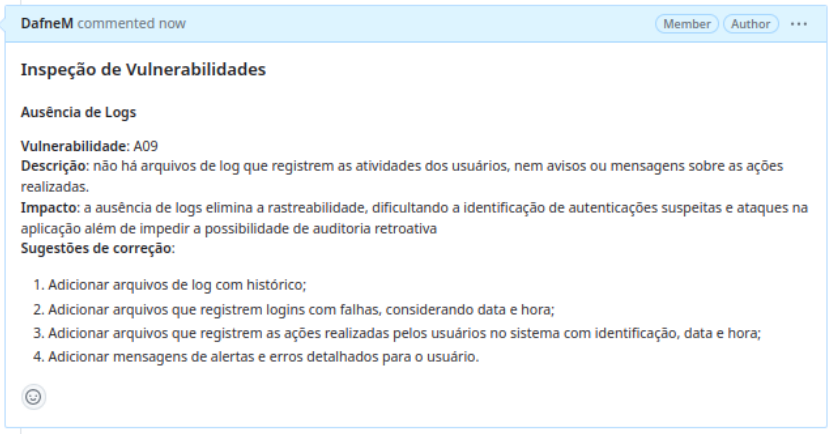
Figura 70 – Ausência de MFA




Fonte: Autores

A ausência de *logs* da aplicação, como na POC1, foi também identificada na inspeção da POC2 (vide Figura 71).

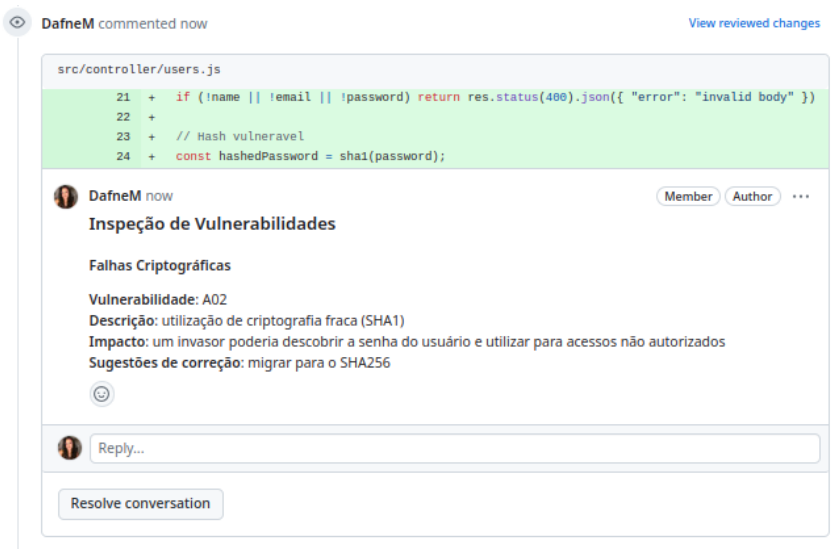
Figura 71 – Relatório de Inspeção de Identificação de Ausência de *Logs* na POC2




Fonte: Autores

A ausência de *logs* impacta diretamente na rastreabilidade da aplicação, como denotado pelo revisor. Outra falha vista na revisão por pares foi a de criptografia fraca, acrescentada também pelos autores, como na Figura 72.

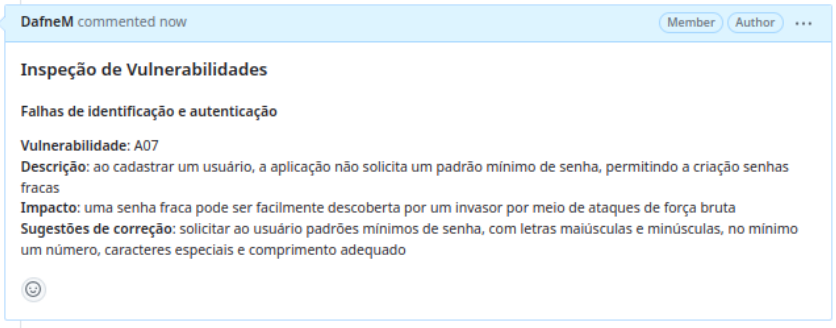
Figura 72 – Relatório de Inspeção de Identificação Identificação de Criptografia fraca na POC2




Fonte: Autores

O SHA1 é um algoritmo obsoleto de criptografia e pode auxiliar um usuário a se passar por outro durante a captura de senha, como comenta o revisor. Além de criptografia fraca, a aplicação permite também a criação de senhas fracas (vide Figura 73).

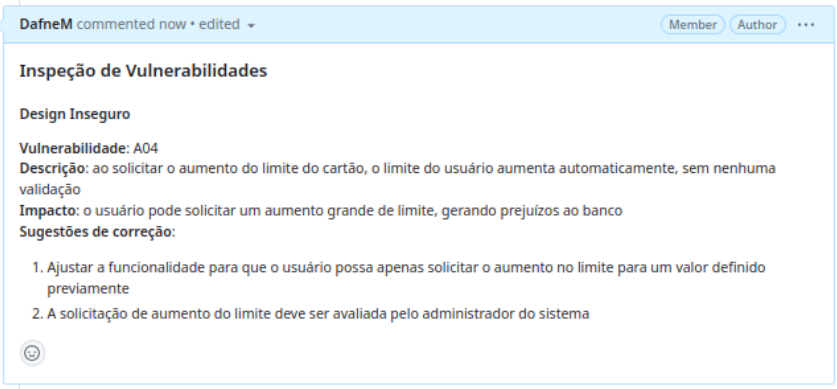
Figura 73 – Relatório de Inspeção de Identificação de Senha fraca na POC2




Fonte: Autores

As senhas fracas podem ser exploradas em ataques de força bruta. Esta vulnerabilidade apareceu durante os Testes de Sistema, assim como a de aumento de limite do cartão, como na Figura 74.

Figura 74 – Relatório de Inspeção de Identificação Aumento de limite do cartão sem verificação na POC2




Fonte: Autores

O usuário solicita o limite e não há tipo de verificação algum. Esta vulnerabilidade é crítica, visto que pode causar prejuízos incalculáveis.

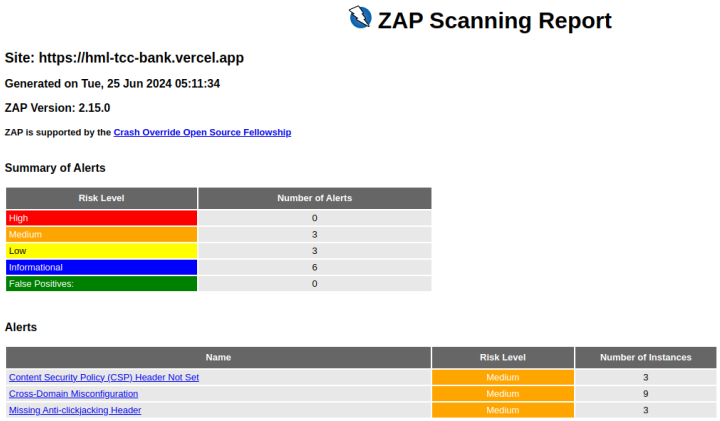
Após a correção das solicitações feitas durante a inspeção, o PR foi mesclado para a Develop, apesar dos alertas de segurança, e puderam ser analisados os Relatórios DAST.

## DAST

### Owasp ZAP

Durante a fase de testes dinâmicos, foram produzidos os relatórios do OWASP ZAP, conforme ilustrado na Figura 75.

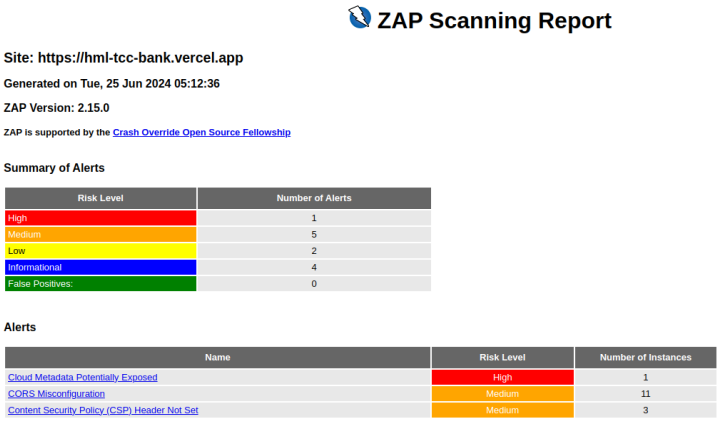
Figura 75 – Resultados *Baseline Scan* ZAP na POC2




Fonte: Autores

O ZAP *Baseline Scan* apresentou três alertas de criticidade média, três de criticidade baixa e seis informacionais. Além da varredura de linha de base, o *Full Scan* apresentou os relatórios da Figura 76

Figura 76 – Resultados *Full Scan* ZAP na POC2



**ZAP Scanning Report**

**Site: https://hml-tcc-bank.vercel.app**

**Generated on Tue, 25 Jun 2024 05:12:36**

**ZAP Version: 2.15.0**

**ZAP is supported by the Crash Override Open Source Fellowship**

**Summary of Alerts**

| Risk Level | Number of Alerts |
|---|---|
| High | 1 |
| Medium | 5 |
| Low | 2 |
| Informational | 4 |
| False Positives: | 0 |

**Alerts**

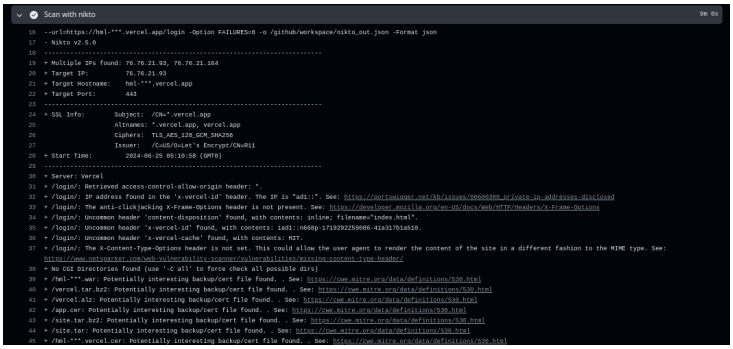
| Name | Risk Level | Number of Instances |
|---|---|---|
| Cloud Metadata Potentially Exposed | High | 1 |
| CORS Misconfiguration | Medium | 11 |
| Content Security Policy (CSP) Header Not Set | Medium | 3 |

Fonte: Autores

O ZAP *Full Scan* alertou os desenvolvedores para uma vulnerabilidade alta, cinco médias, duas baixas e quatro informacionais.

O Nikto apresenta, na própria falha da *action*, os resultados encontrados por ele, conforme a Figura 77.

Figura 77 – Execução da *action* do Nikto na POC2




Fonte: Autores

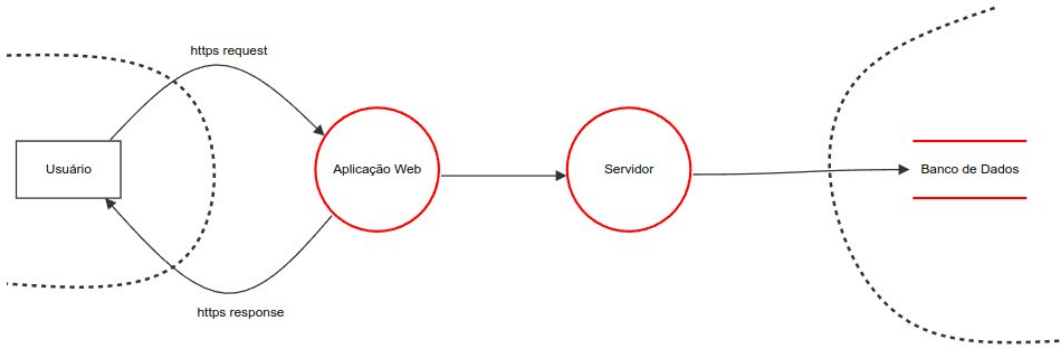
Com a análise dos relatórios e novos Testes de Sistema, o PR para produção foi aceito com as vulnerabilidades que foram corrigidas pelos autores.

### Resolução de Vulnerabilidades

Para a aplicação bancária, também foi utilizado o OWASP *Threat Dragon* para realizar o mapeamento das vulnerabilidades da aplicação. Também foi utilizado o modelo STRIDE.

Na Figura 78, representa-se o diagrama utilizado para a modelagem de ameaças da POC 2.

Figura 78 – Modelagem de Ameaças - Diagrama




Fonte: Autores

A lista de ameaças identificadas para a camada *web* está representada na Figura 79. Foi também incluída uma vulnerabilidade associada à *pipeline* da aplicação.

Figura 79 – Modelagem de Ameaças - Web

**Aplicação Web (Process)**

Camada de interação do usuário

| Number | Title | Type | Priority | Status | Score | Description | Mitigations |
|---|---|---|---|---|---|---|---|
| 1 | XSS em comentários | Tampering | High | Open | | Permite XSS em comentários de investimento | Sanitizar a entrada do usuário para prevenir XSS |
| 2 | Permite senhas fracas | Information Disclosure | Medium | Open | | Permite o uso de senhas fracas | Impor políticas de senhas fortes |
| 3 | Exposição de token na pipeline CI/CD | Information Disclosure | High | Open | | Exposição de token na pipeline CI/CD | Garantir que informações sensíveis não sejam expostas nas pipelines |

Fonte: Autores

A maioria das vulnerabilidades foi implementada no servidor da aplicação, representado pela Figura 80.

Figura 80 – Modelagem de Ameaças - Servidor

**Servidor (Process)**

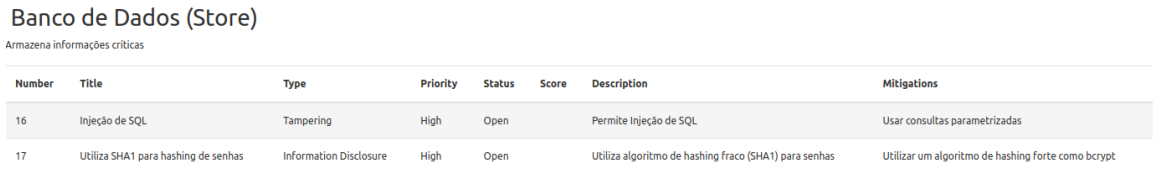
Lida com requisições e interage com o banco de dados

| Number | Title | Type | Priority | Status | Score | Description | Mitigations |
|---|---|---|---|---|---|---|---|
| 4 | Token sem expiração | Information Disclosure | High | Open | | Token não expira | Implementar expiração de token |
| 7 | Sem limite de taxa para chamadas de API | Denial of Service | Medium | Open | | Sem limite de taxa nas chamadas de API | Implementar limite de taxa |
| 8 | Sem cabeçalhos de segurança | Information Disclosure | Medium | Open | | Faltam cabeçalhos de segurança (ex.: CSP, X-Content-Type-Options) | Adicionar cabeçalhos de segurança apropriados |
| 9 | CORS não configurado | Information Disclosure | Medium | Open | | CORS não está configurado | Configurar CORS corretamente |
| 10 | Utiliza dependências desatualizadas | Tampering | High | Open | | A aplicação utiliza dependências desatualizadas | Atualizar regularmente as dependências |
| 11 | Sem MFA | Information Disclosure | High | Open | | Não usa MFA | Implementar MFA |
| 12 | Permite requisições com tokens inválidos | Information Disclosure | High | Open | | Permite requisições mesmo com tokens inválidos | Validar tokens em todas as requisições |
| 13 | Sem logging | Repudiation | Medium | Open | | A aplicação não registra atividades do usuário | Implementar logging |
| 14 | Vulnerabilidade SSRF | Tampering | High | Open | | Permite SSRF (ex.: acesso a arquivos do servidor, LFI) | Validar e sanitizar entradas de URL |
| 15 | SSRF em upload de imagens | Tampering | High | Open | | Permite SSRF ao fazer upload de imagens via URL | Sanitizar e validar URLs para upload de imagens |

Fonte: Autores

As vulnerabilidades associadas ao banco de dados, que estão ligadas à atividade do servidor, estão representadas na Figura 81.

Figura 81 – Modelagem de Ameaças - Banco de Dados



Banco de Dados (Store)

Armazena informações críticas

| Number | Title | Type | Priority | Status | Score | Description | Mitigations |
|---|---|---|---|---|---|---|---|
| 16 | Injeção de SQL | Tampering | High | Open | | Permite Injeção de SQL | Usar consultas parametrizadas |
| 17 | Utiliza SHA1 para hashing de senhas | Information Disclosure | High | Open | | Utiliza algoritmo de hashing fraco (SHA1) para senhas | Utilizar um algoritmo de hashing forte como bcrypt |

Fonte: Autores

As vulnerabilidades foram ordenadas segundo o AVG IMPACTO MEDIO PONDERADO para serem corrigidas.

- A03 - Injeção - 7.15
- A04 – *Design* Inseguro - 6.78
- A10 – Falsificação de Solicitação do Lado do Servidor (SSRF) - 6.72
- A05 – Configuração Incorreta de Segurança - 6.56
- A02 – Falhas Criptográficas - 5.93
- A08 - Falhas de Software e Integridade de Dados: 2.05

As vulnerabilidades foram corrigidas nos seguintes *commits*:

- *Commit* f0a83d7c1dc0650ad8762c0b28999cc84a9f7943[11]: (A03 - Injeção) Correção de *endpoints* relacionados à autenticação do usuário.
- *Commit* 3cec422a8fc472bb38f3fc65e65c0897f77b44b8[12]: (A03 - Injeção) Correção de *endpoints* remanescentes.
- *Commit* 95f38a86ca869f96dfb9607f67ec4c02c3862bb4[13]: (A04 - *Design* Inseguro) Implementação de mecanismo para limitar a utilização de recursos pelo usuário.

[11] Disponível em: <https://github.com/tcc-lucas-dafne/tcc-bank-api/pull/15/commits/f0a83d7c1dc0650ad8762c0b28999cc84a9f7943> e Último acesso em: Julho 2024

[12] Disponível em: <https://github.com/tcc-lucas-dafne/tcc-bank-api/pull/15/commits/3cec422a8fc472bb38f3fc65e65c0897f77b44b8> e Último acesso em: Julho 2024

[13] Disponível em: <https://github.com/tcc-lucas-dafne/tcc-bank-api/pull/12/commits/95f38a86ca869f96dfb9607f67ec4c02c3862bb4> e Último acesso em: Julho 2024

- *Commit* 8e564f63797b7dad4b1b6f1180b586fb4ecbe0ef[14]: (A10 - Falsificação de Solicitação do Lado do Servidor (SSRF)) Utilização de mecanismos de controle para validar URL enviada pelo usuário.

- *Commit* 63434da696c46c836599aa007b145ef2db5f965c[15]: (A10 - Falsificação de Solicitação do Lado do Servidor (SSRF)) Validação do nome do arquivo para impedir que usuário consiga ter acesso a outros diretórios do servidor.

- *Commit* c751b9ab3729635cddb9a9032793f2605e474efa[16]: (A05 - Configuração Incorreta de Segurança) Configuração do CORS, restringindo *origin* que possui acesso à API, além de restringir os métodos aceitos (*GET* e *POST*) e definir os *headers* da aplicação.

- *Commit* 5269ee4a756d1e3fa0dabceed6eed827b67c4956[17]: (A02 - Falhas Criptográficas) Substituição da função de *hash* SHA1, que estava sendo aplicada na senha do usuário, por um mecanismo de proteção mais avançado. Foi utilizada a biblioteca *bcrypt* para essa melhoria.

- *Commit* 60bd17db92f81f4edbca182452688fde0107a9c9[18]: (A08 - Falhas de Software e Integridade de Dados) Removido *token* que estava presente no arquivo de *workflow* do repositório. Foi utilizado um *secret* para armazenamento.

#### 5.4.2.3 Análise de Resultados

O processo de Análise de Resultados da POC2 ocorreu orientando-se pelo Método de Análise de Resultados, descrito no Capítulo 4 - Metodologia. As vulnerabilidades implantadas encontradas pelo *pipeline* estão dispostas no Quadro 7.

[14] Disponível em: <https://github.com/tcc-lucas-dafne/tcc-bank-api/pull/11/commits/8e564f63797b7dad4b1b6f1180b586fb4ecbe0ef> e Último acesso em: Julho 2024
[15] Disponível em: <https://github.com/tcc-lucas-dafne/tcc-bank-api/pull/11/commits/63434da696c46c836599aa007b145ef2db5f965c> e Último acesso em: Julho 2024
[16] Disponível em: <https://github.com/tcc-lucas-dafne/tcc-bank-api/pull/13/commits/c751b9ab3729635cddb9a9032793f2605e474efa> e Último acesso em: Julho 2024
[17] Disponível em: <https://github.com/tcc-lucas-dafne/tcc-bank-api/pull/14/commits/5269ee4a756d1e3fa0dabceed6eed827b67c4956> e Último acesso em: Julho 2024
[18] Disponível em: <https://github.com/tcc-lucas-dafne/tcc-bank/pull/11/commits/60bd17db92f81f4edbca182452688fde0107a9c9> e Último acesso em: Julho 2024

Quadro 7 – Vulnerabilidades implantadas encontradas pela *pipeline*

| OWASP | ID | Vulnerabilidade | Identificada |
|---|---|---|---|
| A01 | P2-001-A01 | Token sem expiração | |
| A01 | P2-002-A01 | Não existe nenhum *rate limit* nas chamadas da API | X |
| A02 | P2-003-A02 | Uso de criptografia fraca (SHA1) | X |
| A03 | P2-004-A03 | *SQL Injection* | X |
| A03 | P2-005-A03 | *XSS Injection* | |
| A04 | P2-006-A04 | Aumento de limite de cartão sem verificação | X |
| A05 | P2-007-A05 | Não há *headers* de segurança na aplicação | X |
| A05 | P2-008-A05 | Ausência de configuração de CORS | X |
| A07 | P2-009-A07 | Não é utilizado MFA | X |
| A07 | P2-010-A07 | Ausência de identificação única | X |
| A07 | P2-011-A07 | Ausência de padrão mínimo de senha | X |
| A07 | P2-012-A07 | Token JWT da sessão não é corretamente invalidado | |
| A07 | P2-013-A07 | Permissão de requisição mesmo com expiração de token | |
| A08 | P2-014-A08 | Exposição de token na pipeline CI/CD | X |
| A09 | P2-015-A09 | Ausência de Logs | X |
| A10 | P2-016-A10 | URL executada no servidor | X |
| A10 | P2-017-A10 | *LFI* | |

Fonte: Autoria própria.

Das duas vulnerabilidades A01 da aplicação, apenas a P2-002-A01 foi observada. Sobre a injeção de código, a Injeção de SQL foi reconhecida, mas a XSS não foi. Duas das cinco vulnerabilidades A07 foram identificadas. O *pipeline* percebeu todas as vulnerabilidades categorizadas como A02, A04, A05, A08 e A09. Por último, 50% das vulnerabilidades A010 foram notadas. Como o *pipeline* proposto é subdividido em etapas, as vulnerabilidades foram segmentadas por etapa de detecção, sendo vistas no Quadro 8.

Quadro 8 – Vulnerabilidades encontradas pela *pipeline*

| Vulnerabilidade | Auditoria | Dependências | SAST | Inspeção/ Testes de Sistema | DAST |
|---|---|---|---|---|---|
| P2-001-A01 | | | | | |
| P2-002-A01 | | | X | | |
| P2-003-A02 | | | | X | |
| P2-004-A03 | | | X | | |
| P2-005-A03 | | | | | |
| P2-006-A04 | | | | X | |
| P2-007-A05 | | | | | X |
| P2-008-A05 | | | | | X |
| P2-009-A07 | | | | X | |
| P2-010-A07 | | | | X | |
| P2-011-A07 | | | X | | |
| P2-012-A07 | | | | | |
| P2-013-A07 | | | | | |
| P2-014-A08 | X | | | | |
| P2-015-A09 | | | | X | |
| P2-016-A10 | | | | | X |
| P2-017-A10 | | | | | |

Fonte: Autoria própria.

A maior parte das falhas de segurança da POC2 foi identificada pelas etapas de SAST, Inspeção/Testes de Sistema e DAST. Para o aprofundamento das ferramentas responsáveis pela detecção, as vulnerabilidades encontradas pelo *pipeline* na POC2 foram especificadas por métodos de identificação, conforme consta no Quadro 6.

Quadro 9 – Vulnerabilidades encontradas pela *pipeline*

| OWASP TOP 10 | Vulnerabilidade | Método de identificação |
|---|---|---|
| A01 | P2-001-A01 | - |
| A01 | P2-002-A01 | CodeQL |
| A02 | P2-003-A02 | Inspeção |
| A03 | P2-004-A03 | SonarCloud |
| A03 | P2-005-A03 | - |
| A04 | P2-006-A04 | Testes de sistema |
| A05 | P2-007-A05 | OWASP ZAP |
| A05 | P2-008-A05 | OWASP ZAP |
| A07 | P2-009-A07 | Testes de sistema |
| A07 | P2-010-A07 | Testes de sistema |
| A07 | P2-011-A07 | Inspeção/Testes de sistema |
| A07 | P2-012-A07 | - |
| A07 | P2-013-A07 | - |
| A08 | P2-014-A08 | GitGuardian |
| A09 | P2-015-A09 | Inspeção |
| A10 | P2-016-A10 | OWASP ZAP |
| A10 | P2-017-A10 | - |

Fonte: Autoria própria.

Assim, a quantidade total de vulnerabilidades implantadas foi 17. Destas, o *pipeline* proposto nesta monografia, verificou 12 e não verificou 5, resultando na métrica 7 e no resultado final (Equação 8).

$$\text{taxa de vulnerabilidades verificadas} = \frac{12}{17} \tag{7}$$

$$\text{taxa de vulnerabilidades verificadas} \approx 0{,}71 \tag{8}$$

Ao se aproximar de 1, esta métrica revela que a maioria das vulnerabilidades implantadas foi identificada, sendo o *pipeline* considerado eficaz para esta Prova de Conceito 2.

Além das vulnerabilidades propositalmente implantadas, as ferramentas identificaram vulnerabilidades no código que ocorreram involuntariamente. Assim, a Tabela 4 demonstra as vulnerabilidades adicionais encontradas.

Tabela 4 – Vulnerabilidades extras na POC2

| **Repositório** | **Total Encontradas** | **Verificadas** | **Extras** |
|---|---|---|---|
| Frontend | 177 | 10 | 167 |
| Backend | 2 | 2 | 0 |
| Total | 179 | 12 | 167 |

Foram identificados seis falsos positivos no GitGuardian que não foram contabilizados. No *Backend*, não há vulnerabilidades. As Equações 9 e 10 apresentam a quantidade de vulnerabilidades percebidas pelo pipeline e extras, respectivamente.

$$\text{total de vulnerabilidades percebidas pelo } \textit{pipeline} \approx 179 \tag{9}$$

$$\text{vulnerabilidades extras} \approx 167 \tag{10}$$

Baseado na taxa de vulnerabilidades verificadas e na quantidade de vulnerabilidades extras identificadas, o *pipeline* alcançou o objetivo indicado para a Prova de Conceito 2.

## 5.5 Considerações Finais do Capítulo

Este capítulo detalhou o *pipeline* DevSecOps desenvolvido nas POCs. Inicialmente, é apresentada a CONTEXTUALIZAÇÃO que embasou a presente proposta. Sequencialmente, é apresentado o *pipeline* DevSecOps desenvolvido. Depois, são descritas as PROVAS DE CONCEITO. Para cada Prova de Conceito, há o Objeto de Estudo, as Vulnerabilidades exploráveis na aplicação, com insumos acerca de cada vulnerabilidade, apresentando o *pipeline* DevSecOps aplicado e as vulnerabilidades corrigidas.

# 6 Análise de Resultados

## 6.1 Considerações Iniciais do Capítulo

Este capítulo discorre sobre a análise de resultados do trabalho, estabelecendo aspectos das duas Provas de Conceito desenvolvidas, comparando aspectos quantitativos, em RESULTADOS, como: VULNERABILIDADES IDENTIFICADAS, PERFIL DE RISCO CRÍTICO, TOP TIPOS DE VULNERABILIDADES, RUNTIME. Adicionalmente, ponderando aspectos qualitativos na ANÁLISE QUALITATIVA. Por fim, têm-se as CONSIDERAÇÕES FINAIS DO CAPÍTULO.

## 6.2 Resultados

A análise consolidada com os dados derivados das Provas de Conceito 1 e 2 ocorreu de acordo com o Método de Análise de Resultados, descrito no Capítulo 4 - Metodologia. A seguir, são apresentados os desdobramentos decorrentes das identificações de vulnerabilidades.

### 6.2.1 Vulnerabilidades Identificadas

Acerca das vulnerabilidade incorporadas pelos autores nas Provas de Conceito, a Tabela 5 revela resumidamente os resultados quantitativos de vulnerabilidades verificadas e a taxa de vulnerabilidades verificadas para cada Prova de Conceito.

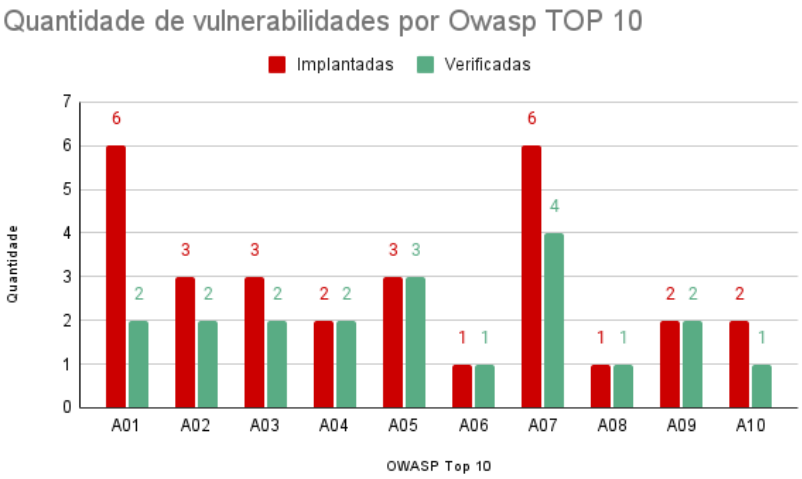
Tabela 5 – Vulnerabilidades verificadas pelo *Pipeline*

| **Prova de Conceito** | **Verificadas** | **Incorporadas** | **Taxa** |
|---|---|---|---|
| POC1 | 8 | 12 | 0,67 |
| POC2 | 12 | 17 | 0,71 |

Fonte: Autoria própria.

É possível fazer a análise por meio da Figura 82, que demonstra o gráfico das vulnerabilidades implantadas e verificadas, classificadas para cada um dos Top 10 Tipos de Vulnerabilidades OWASP.

Figura 82 – Gráfico de Vulnerabilidades por Categoria




Fonte: Autores

A vulnerabilidade com a menor quantidade de detecções em relação ao total é a A01. Isso ocorre visto que os autores selecionaram a ferramenta OWASP ZAP para fazer estas verificações, uma vez que a sua documentação inclui estes tipos de vulnerabilidades. No entanto, após realizarem pesquisas, entraram em contato com a comunidade do ZAP e verificaram, como apresentado na seção 7.5, que não há como fazer testes de controle de acesso no CI/CD do ZAP até o momento da escrita desta monografia.

Calculando a soma das POCs 1 e 2, do total de 29 incorporadas, o *pipeline* verificou 20 vulnerabilidades. Assim, a taxa de verificadas resultante foi, conforme constam nas Equações (11) e (12):

$$\text{taxa de vulnerabilidades verificadas} = \frac{20}{29} \tag{11}$$

$$\text{taxa de vulnerabilidades verificadas} \approx 0,69 \tag{12}$$

A métrica final resultante, para o caso da taxa de vulnerabilidades verificadas, ao considerar ambas as POCs, aproximou-se de 1.

A respeito das fraquezas de segurança extras reconhecidas, o Quadro 5 resume o número de vulnerabilidades extras segmentadas por Prova de Conceito.

Tabela 6 – Vulnerabilidades extras encontradas pelo *Pipeline* em cada POC

| Prova de Conceito | Quantidade Extras |
|---|---|
| POC1 | 179 |
| POC2 | 167 |

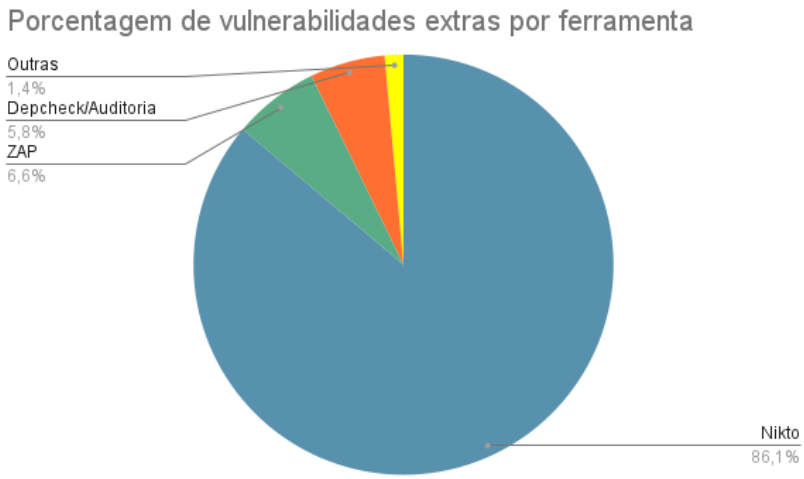
Fonte: Autoria própria.

A quantidade de vulnerabilidades extras total está especificada na Equação (13).

$$\text{vulnerabilidades extras} \approx 346 \tag{13}$$

A porcentagem aproximada de vulnerabilidades extras, separada por ferramentas, consta ilustrada na Figura 83, que demonstra o gráfico de vulnerabilidades extras por ferramenta.

Figura 83 – Vulnerabilidades extras por ferramentas




Fonte: Autores

O Nikto é a ferramenta que mais identificou vulnerabilidades extras, sendo responsável por 86.1% da identificação, seguido do OWASP ZAP com 6.6%, e da checagem de dependências, com 5.8%.

No total, foram verificadas 20 vulnerabilidades principais e 346 adicionais, totalizando 366 vulnerabilidades identificadas pelo *pipeline*, comprovando solução eficaz.

### 6.2.2 Perfil de Risco Crítico

A métrica de Perfil de Risco Crítico definida na Capítulo 4 - Metodologia considera o fator de Impacto Médio Ponderado na ordenação.

Primeiro, as vulnerabilidades da POC1 foram ordenadas de acordo com esta métrica:

1. A03 - Injeção - 7.15
2. A04 – *Design* Inseguro - 6.78
3. A07 – Falhas de Identificação e Autenticação - 6.50
4. A01 – Quebra de Controle de Acesso - 5.93
5. A09 – Falhas de Registro e Monitoramento de Segurança - 4.99
6. A06 - Componentes Vulneráveis - 2.2

Na POC2, as vulnerabilidades também foram ordenadas pelo Perfil de Risco Crítico.

1. A03 - Injeção - 7.15
2. A08 – Falhas de Software e Integridade de Dados: 2.05
3. A04 – *Design* Inseguro - 6.78
4. A10 – Falsificação de Solicitação do Lado do Servidor (SSRF) - 6.72
5. A05 – Configuração Incorreta de Segurança - 6.56
6. A02 – Falhas Criptográficas - 5.93

As vulnerabilidades foram corrigidas de acordo com a priorização baseada no Perfil de Risco Crítico, evidenciando a eficácia do *pipeline*. Um indicador de qualidade é a minimização das vulnerabilidades críticas. Esta minimização foi alcançada, pois após a correção, considerando aquelas propositalmente introduzidas pelos autores, não há vulnerabilidades de alto risco nas POCs.

### 6.2.3 Top Tipos de Vulnerabilidades

A métrica Top Tipos de Vulnerabilidades considera o Fator de Taxa Média de Incidência para a ordenação. As vulnerabilidades da POC1 foram ordenadas conforme este fator:

1. A06 - Componentes Vulneráveis (8.77%)
2. A09 – Falhas de Registro e Monitoramento de Segurança (6.51%)

3. A01 – Quebra de Controle de Acesso (3.81%)
4. A03 - Injeção (3.37%)
5. A07 – Falhas de Identificação e Autenticação (2.55%)

Na POC2, esta ordenação também foi realizada:

1. A09 – Falhas de registro e monitoramento de segurança (6.51%)
2. A05 – Configuração Incorreta de Segurança (4.51%)
3. A02 – Falhas Criptográficas (4.49%)
4. A03 - Injeção (3.37%)
5. A04 – *Design* Inseguro (3.00%)
6. A10 – Falsificação de Solicitação do Lado do Servidor (2.72%)
7. A08 – Falhas de Software e Integridade de Dados (2.05%)

As vulnerabilidades identificadas possuem um plano de mitigação, e foram tratadas conforme esse plano. Segundo Prates (2019), um bom indicador de segurança é ter o menor número possível de vulnerabilidades sem um plano de mitigação. Nesse sentido, o indicador está satisfatório, pois as vulnerabilidades introduzidas pelos autores foram mitigadas de acordo com o plano estabelecido.

### 6.2.4 *Runtime*

O tempo de execução do *pipeline* varia a depender da quantidade de vulnerabilidades existentes na aplicação. À medida que o número de vulnerabilidades aumenta, o tempo que as ferramentas levam para realizar a varredura completa e identificar todos os pontos críticos tende a crescer. Os autores adotaram estratégias de paralelização de fluxos de trabalho, separando em *jobs* distintos *actions* que podem ser executadas concorrentemente para a otimização do tempo.

Considerando as Provas de Conceito deste trabalho, o tempo médio do *pipeline* é por volta de 22 minutos. Os parâmetros das ferramentas de segurança alteram o tempo de execução do *pipeline* e podem ser modificados segundo a necessidade de cada projeto. É o caso do *–nvdApiKey {{secrets.*NVD_API_KEY}} do OWASP *Dependency Check*, que utiliza a API do NVD para consulta; o -T do ZAP, que define o tempo de varredura da ferramenta, e o -Option FAILURES=0 do Nikto, que indica o máximo de falhas até a finalização da execução

O *pipeline* foi dividido conceitualmente em três fases para melhor compreensão do seu tempo de execução. A fase de abertura do *Pull Request* para a *Develop*, a fase de *Deploy* para homologação/testes DAST, e a fase de *Deploy* para a produção/geração de *release*.

### *Pull Request* para a Develop

Na fase de abertura do *Pull Request* para a *develop*, o *workflow* do CodeQL tem uma duração que varia de 1 minuto a 1 minuto e 30 segundos, aproximadamente, em cada repositório. O *workflow build_sast.yaml* que contém o build da aplicação; o *npm-audit* e a análise do SonarCloud exibem comportamentos análogos nos repositórios das duas Provas de Conceito, e também demandam cerca de 1 minuto para conclusão.

O comportamento do *workflow* de verificação de dependências vulneráveis varia em cada uma das Provas de Conceito, levando de 1 minuto até 12 minutos, aproximadamente, para executar. Esta variação de tempo ocorre devido à oscilação temporal da ferramenta OWASP *Dependency Check*, que acompanha o número de falhas em dependências presentes na aplicação. Na tabela 7, é apresentado o tempo de execução do *Workflow depcheck.yaml*.

Tabela 7 – Tempo de execução do *Workflow depcheck.yaml*

| **POC** | **Repositório** | ***Runtime*** |
|---|---|---|
| POC1 | *Frontend* | ≈6min |
| POC1 | *Backend* | ≈1min |
| POC2 | *Frontend* | ≈12min |
| POC2 | *Backend* | ≈1min |

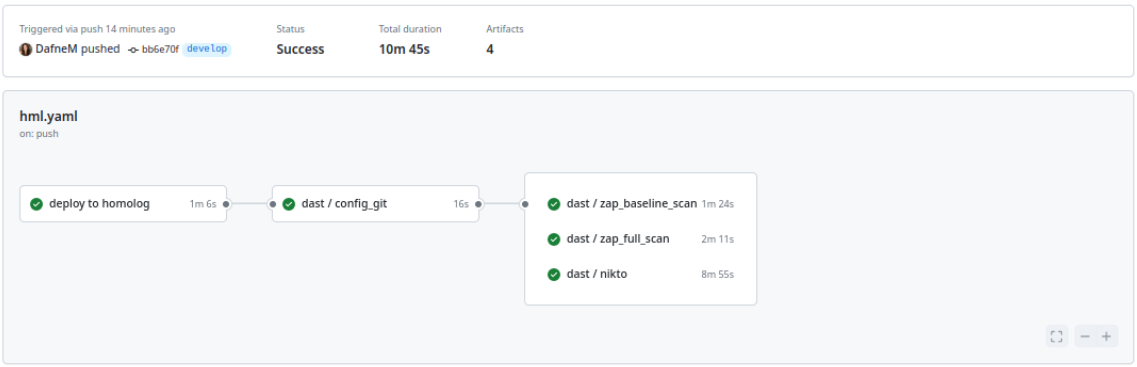
Fonte: Autoria própria.

Como os *jobs* atuam de forma concorrente, a média de tempo total entre a abertura do *Pull Request* e a finalização dos testes estáticos foi aproximadamente 9 minutos. Deve ser acrescentado a esta média, o tempo até a concepção do Relatório de Vulnerabilidades por Inspeção no *pair review*.

### *Deploy* para Homologação

Na fase de *deploy* para homologação e testes DAST, o *runtime* dos fluxos de trabalho variou entre 10 e 12 minutos nos repositórios *Frontend*. O tempo dos fluxos de trabalho da POC1 no *Frontend* foi aproximadamente 10m e 45s, conforme a Figura 84.

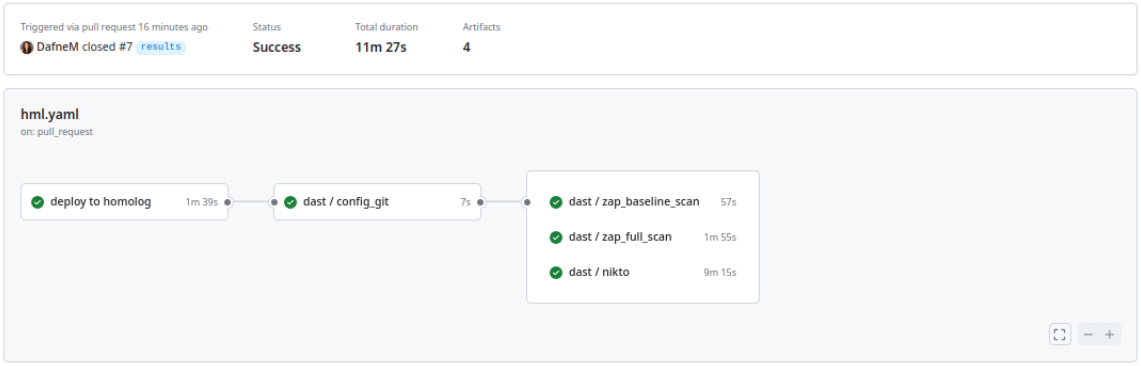
Figura 84 – Tempo de *Deploy* para Homologação e testes DAST no *Frontend* da POC1




Fonte: Autores

A fase mais demorada desta etapa é a fase de testes. A Figura 85 apresenta o tempo do *workflow* na POC2.

Figura 85 – Tempo de *Deploy* para Homologação e testes DAST no *Frontend* da POC2




Fonte: Autores

O *runtime* do fluxo de trabalho no *Frontend* desta POC2 foi 11m e 27s. Cada teste da ferramenta ZAP executou entre cerca de 1 minuto e 2 minutos. O Nikto é o principal responsável pelo tempo nos testes DAST, executando, na POC1, em 8 minutos e 55 segundos, e na POC2, em 9 minutos e 15 segundos. Este período de tempo deve-se à configuração da quantidade máxima de falhas que foi desabilitada.

A média de tempo total entre a aprovação do *Pull Request* e a finalização dos testes dinâmicos foi por volta de 9 minutos 45 segundos. Devido à natureza ativa dos testes DAST, os autores decidiram mantê-la por este período, considerando que isso não afeta significativamente, uma vez que o ambiente de homologação é precisamente utilizado para testes.

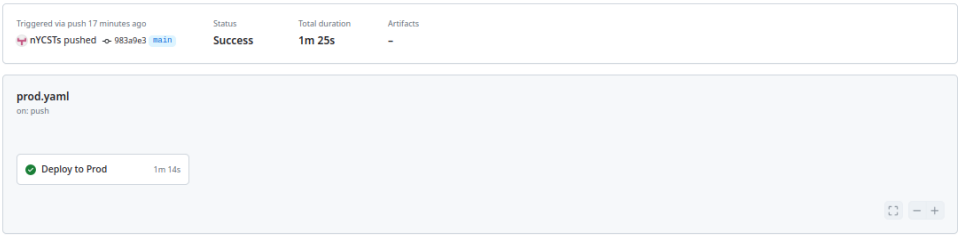
No *Backend*, não há a fase de DAST. Isso faz com que a etapa seja mais rápida. A duração máxima para a conclusão do *Deploy* foi 1 minuto.

A etapa de *Deploy* da aplicação para o ambiente de homologação no *Backend*, por não realizar testes DAST, demorou menos de um minuto.

### *Deploy* para Produção

A fase de *Deploy* para produção levou no máximo 2 minutos para executar. A Figura 86 mostra um exemplo de execução do *workflow* do *Frontend* da POC1 no Github.

Figura 86 – Tempo de *Deploy* em Produção do *Frontend* da POC1

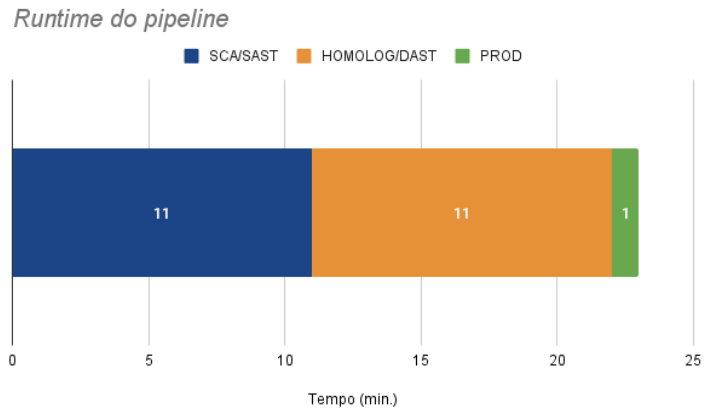



Fonte: Autores

Esta etapa durou 1m e 25s para ser realizada a partir do momento do aceite do PR até a finalização de todos os *steps*.

A Figura 87 demonstra o gráfico de barras empilhadas que apresenta, em minutos, o tempo gasto em cada etapa do *Pipeline* em relação ao tempo total. Este gráfico desconsidera o tempo de Inspeção de Código e Testes de Sistema.

Figura 87 – Tempo de execução do *pipeline* por etapas




Fonte: Autores

A etapa de DAST é a que dispõe de maior tempo, seguida da SAST/SCA que compreende toda a auditoria, checagem de dependências, SCA e SAST, e por último a etapa de *Deploy* para produção.

### 6.2.5 Análise Qualitativa

No *pipelne* DevSecOps proposto, a integração e a entrega contínuas são realizadas de forma automatizada, mantendo os procedimentos consolidados na ferramenta *GitHub*. A aplicação da segurança é feita por meio de *actions* que são disparadas por *triggers*, também automatizadas. O *pipeline*, no contexto de integração, consolidação de informações e automatização, classifica-se em 5, de acordo com a escala Likerti.

Quanto aos relatórios gerados pelo *pipeline*, são exibidos detalhamentos das falhas de segurança, impacto e opções corretivas. Algumas ferramentas como o SonarCloud e o GitGuardian indicam o caminho e o trecho do código para o *Security Hotspot* e para as vulnerabilidades. Outras como o OWASP ZAP e OWASP *Dependency Check* fazem correlações das vulnerabilidades identificadas com a OWASP, os CWEs e os CVEs conhecidos pela comunidade. Na escala Likerti, o detalhamento dos relatórios classifica-se também em 5.

Na primeira fase das POCs, com a inserção das vulnerabilidades, de acordo com a escala Likerti, o código não satisfazia as diretrizes OWASP. Com a solução proposta e as correções, o código passou então a atender/satisfazer as diretrizes OWASP.

Finalmente, considerando as análises quantitativas e qualitativas, o *pipeline* obteve resultados positivos nas Provas de Conceito.

### 6.2.6 Contribuições do *Pipeline* em Comunidades *Open Source*

Além das Provas de Conceito, este trabalho contribuiu para projetos *Open Source* com a realização testes de se/gurança no SIGE (*Energy Management System*).O SIGE é um projeto da Universidade de Brasília responsável pelo gerenciamento de energia. O repositório pode ser acessado no *link*: <https://gitlab.com/lappis-unb/projetos-energia/SIGE/sige-master>.

Para fazer os testes, os autores desse trabalho solicitaram a aprovação do mantenedor responsável Renato Coral, que concordou com os testes a serem realizados.

Como solicitado pelo mantenedor, os esforços foram concentrados no repositório recentemente refatorado "SIGE-Master". Os autores fizeram um *fork* do repositório que pode ser acessado pelo *link*: <https://gitlab.com/tcc-dafne/sige-master>.

O projeto está hospedado no Gitlab, e por isto, algumas alterações foram realizadas para viabilizar a execução do *pipeline* nesta ferramenta. No Gitlab, não são utilizadas *actions* e um conjunto de arquivos *yaml*. Pelo contrário, apenas um arquivo é responsável por toda a configuração da operação, sendo ele o *.gitlab-ci.yml*. Os autores optaram por utilizar o *docker* para facilitar a configuração dos ambientes.

A linguagem principal do projeto é o *Python*. Então, outras adaptações precisaram ser feitas. A primeira delas foi na etapa de auditoria/checagem de dependências, que não utiliza mais o *npm-audit*, o *owasp-dependency-check*, o *snyk* e nem o dependabot. Neste momento, optou-se pela utilização do *pip-audit* e do *safety* em sua versão *open source*. O Código 14 demonstra isso.

Código 14 – Código do *pip-audit*

```
pip-audit:
    ...
  image: python:latest
  before_script:
    - pip install pip-audit
    - pip install -r requirements.txt
  script:
    - pip-audit -r requirements.txt --format=json > pip-audit.json
  allow_failure: true
    ...

artifacts:
paths:
  - pip-audit.json
when: always
```

Fonte: Autoria própria.

Com a execução do *pip-audit*, foi possível verificar as dependências vulneráveis, como consta na Figura 88

Figura 88 – Resultados do *pip-audit* no SIGE


Fonte: Autores

Foi encontrada uma dependência vulnerável no Django. Para auxiliar na etapa de verificação de dependências, o *safety* foi escolhido (vide Código 15).

Código 15 – Código do *safety*

```
safety-check:
  ...
  image: python:latest
  before_script:
    - pip install virtualenv
    - virtualenv venv
    - source venv/bin/activate
    - pip install safety
    - pip install -r requirements.txt
  script:
    - safety scan -r requirements.txt --full-report > safety_out.json
  ...

  artifacts:
    paths:
      - safety_out.json
    when: always
```

Fonte: Autoria própria

Pelo *safety* não foram encontradas vulnerabilidades, mas 16 vulnerabilidades foram ignoradas[1] pela ferramenta. Os pacotes considerados como "vulnerabilidades ignoradas"foram *werkzeug*, *djangorestframework-simplejwt* e *Django*. Para o DAST, foi mantida a ferramenta OWASP ZAP. O Código 16 representa as etapas de implementação desta ferramenta.

Código 16 – Código *Upload* OWASP ZAP

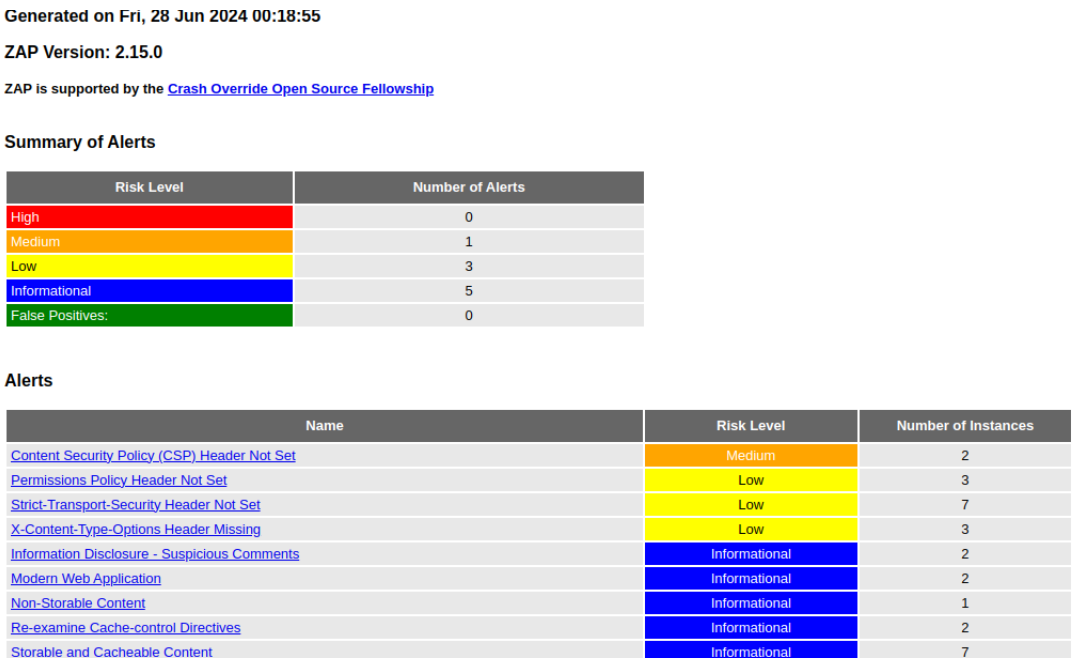
```
zap:
    ...
  script:
    - docker pull ghcr.io/zaproxy/zaproxy:stable
    - docker pull zaproxy/zap-stable
    - docker run -v $CI_PROJECT_DIR:/zap/wrk/:rw -t ghcr.io/zaproxy/zaproxy:stable
    ↪  zap-baseline.py -t $URL_BACK -r report.html
    ...

  artifacts:
    paths:
      - report.html
    when: always
```

Fonte: Autoria própria

[1] por ignoradas, entende-se por categorizadas como "vulnerabilidades ignoradas"pela ferramenta

O relatório gerado pelo OWASP ZAP encontra-se na Figura 89.

Figura 89 – Resultados do OWASP Zap no SIGE



**Generated on Fri, 28 Jun 2024 00:18:55**

**ZAP Version: 2.15.0**

**ZAP is supported by the Crash Override Open Source Fellowship**

**Summary of Alerts**

| Risk Level | Number of Alerts |
|---|---|
| High | 0 |
| Medium | 1 |
| Low | 3 |
| Informational | 5 |
| False Positives: | 0 |

**Alerts**

| Name | Risk Level | Number of Instances |
|---|---|---|
| Content Security Policy (CSP) Header Not Set | Medium | 2 |
| Permissions Policy Header Not Set | Low | 3 |
| Strict-Transport-Security Header Not Set | Low | 7 |
| X-Content-Type-Options Header Missing | Low | 3 |
| Information Disclosure - Suspicious Comments | Informational | 2 |
| Modern Web Application | Informational | 2 |
| Non-Storable Content | Informational | 1 |
| Re-examine Cache-control Directives | Informational | 2 |
| Storable and Cacheable Content | Informational | 7 |

Fonte: Autores

Não foi encontrado nenhum alerta de criticidade alta algum, mas foram encontrados um alerta de média, três de baixa criticidade e cinco apenas informacionais. Adicionalmente, o NIkto também foi mantido para auxiliar os testes dinâmicos, como pode ser visualizado no Código 17.

Código 17 – Código do Nikto

```
1  nikto:
2    ...
3     script:
4       - docker pull hysnsec/nikto
5       - docker run --rm -v $CI_PROJECT_DIR:/tmp hysnsec/nikto -h $URL_BACK -o
       ↪  /tmp/nikto_out.html -Format html
6       - ls $CI_PROJECT_DIR
7       - find . -name nikto_out.html
8    ...
9
10    artifacts:
11      paths:
12        - nikto_out.html
13      when: always
```

Fonte: Autoria própria

O Código do Nikto gera o relatório da Figura 90.

Figura 90 – Resultados do Nikto no SIGE


Fonte: Autores

O Nikto testou um host; levou aproximadamente 217 segundos para executar, e encontrou vinte erros no total. Todos os testes falham ao encontrar pelo menos uma vulnerabilidade.

Estes relatórios detalhados podem ser utilizados como base para perceber e realizar melhorias no Master do SIGE nas novas versões disponibilizadas.

Alguns planos pagos do Gitlab permitem habilitar o painel de segurança com análises SAST, DAST, Análises de Segredos e de dependências.

## 6.3 Considerações Finais do Capítulo

Este capítulo abordou a Análise de Resultados deste trabalho com elementos das duas POCs, relacionando dados quantitativos de Vulnerabilidades Identificadas, Densidade de Defeitos, Perfil de Risco Crítico, Top Tipos de Vulnerabilidade, *Runtime*, além de dados qualitativos na Análise Qualitativa e Contribuições do *Pipeline* em Comunidades

# 7 Conclusão

## 7.1 Considerações Iniciais do Capítulo

Este capítulo oferece uma visão do trabalho relacionado pelos autores, iniciando-se pelas QUESTÕES DE PESQUISA e DESENVOLVIMENTO, respondidas ao finalizar o trabalho; OBJETIVOS GERAL E ESPECÍFICOS DO TRABALHO alcançados; seguido dos TRABALHOS FUTUROS. Por fim, têm-se as CONSIDERAÇÕES FINAIS DO CAPÍTULO.

## 7.2 Questões de Pesquisa e Desenvolvimento

No Capítulo de Introdução, foram acordados alguns questionamentos. Com o término do trabalho, os mesmos podem ser respondidos, conforme colocado a seguir:

- Questão_de_Pesquisa_01: Quais são as principais práticas de cibersegurança recomendadas pela comunidade especializada?

  Conforme consta no Capítulo 2 - Referencial Teórico, mais especificamente na seção 2.3 OWASP, há menção as principais práticas de cibersegurança recomendas pela comunidade, com destaque para as OWASP Top Ten.

- Questão_de_Pesquisa_02: Quais são as principais ferramentas *open source* que viabilizam testar as práticas de cibersegurança antes mesmo da implantação do software?

  Conforme consta no Capítulo 3 - Suporte Tecnológico, mais especificamente na seção 3.3 Ferramentas de Apoio ao *Pipeline* DevSecOps, há menção às ferramentas escolhidas (open source e com versão de uso gratuíta): Vercel, Heroku, SAST, SCA e DAST. Adicionalmente, o capítulo acorda outras ferramentas que apoiam o desenvolvimento como um todo, cabendo menção: GitHub, Material UI, Bootstrap, entre outras.

- Questão_de_Desenvolvimento: Como essas práticas de cibersegurança podem ser incluídas na fase de desenvolvimento de um software, e testadas usando ferramentas open source?

  Conforme consta detalhado ao longo dos Capítulos 5 e 6 dessa monografia, há descritivo sobre o *Pipeline* DevSecOps desenvolvido pelos autores, com uso dos insumos

do Referencial Teórico e apoio dos ferramentais do Suporte Tecnológico; além do descritivo da Análise de Resultados, submetendo o *pipeline* às ferramentas de verificação, com métricas, e método de análise de abordagem quantitativa e qualitativa.

## 7.3 Objetivos Geral e Específicos

As atividades e os subprocessos relacionados ao TCC, descritas na subseção Atividades e Subprocessos - Primeira Etapa do TCC e na subseção Atividades e Subprocessos - Segunda Etapa do TCC foram concluídos, ficando pendente apenas a apresentação para a banca. Foram alcançados e/ou direcionados os seguintes objetivos específicos definidos no Capítulo 1 - Introdução:

- Estudo sobre métodos ágeis, mais especificamente no que compreende o surgimento do DevOps. É importante realizar esse estudo investigativo, junto à comunidade especializada, uma vez que ao se orientar por cibersegurança, o *modus operandi* desses métodos, considerando Desenvolvimento e Operações, não pode ser prejudicado. Nesse caso, o objetivo foi alcançado. Insumos Gerados: Contextualização, no Capítulo 1 - Introdução; Apontamentos Gerais em DevOps, no Capítulo 2 - Referencial Teórico e Método de Desenvolvimento, no Capítulo 4 - Metodologia;

- Levantamento sobre práticas de cibersegurança. É importante realizar esse levantamento, ainda junto à comunidade especializada, para compreender não apenas quais são essas práticas, mas também seus benefícios e suas preocupações quando incorporadas à fase de desenvolvimento. Nesse caso, o objetivo foi alcançado. Insumos Gerados: Segurança de Software, OWASP e DevSecOps, no Capítulo 2 - Referencial Teórico;

- Levantamento de ferramentas, de preferência *open source*, para utilizar no *pipeline* desenvolvimento/teste. É importante realizar esse levantamento, ainda junto à comunidade especializada, para identificar ferramentas capazes de detectar diversos tipos de vulnerabilidades antes mesmo da implantação do software em si. Nesse caso, o objetivo foi alcançado. Insumos Gerados: Capítulo 3 - Suporte Tecnológico, e

- Implementação de cenários de uso em um software, orientado-se pelo estudo e pelos levantamentos realizados anteriormente. É importante realizar essa implementação, no intuito de conferir insumos de cunho mais prático ao *pipeline* DevSecOps. Entende-se por cenários de uso, nesse caso, e fazendo um adaptação com base em Silva e Alves (2018), compreendendo a especificação de situações pontuais, vistas como problemas de cibersegurança, e que demandam ser tratadas no software. Nesse

caso, o objetivo foi alcançado. Insumos Gerados: Descritivo das Provas de Conceito sobre como o estudo foi conduzido, desde à definição de objetivos, passando pela identificação de requisitos técnicos e pela implementação das PoCs, até a revelação dos resultados, que trata de forma conjunta teste & validação e análise de resultados. Insumos Gerados: Capítulo 5 - Pipeline DevSecOps.

- Análise dos resultados obtidos visando reportar cada aspecto observado ao longo do processo. Utilizando uma abordagem híbrida: ora quantitativa, com revelação de quantas vulnerabilidade foram identificadas orientando-se pelo *pipeline* DevSecOps; ora qualitativa, com revelação de benefícios, preocupações, dentre outros aspectos não quantificáveis. Nesse caso, o objetivo foi alcançado. Insumos Gerados: Capítulo 6 - Análise de Resultados.

Ao alcançar todos os Objetivos Específicos, foi atingido também o Objetivo Geral que é: demonstração, utilizando Provas de Conceito, de como práticas de cibersegurança podem ser incluídas na fase de desenvolvimento e testadas via ferramentas *open source*, antes mesmo da implantação do software.

## 7.4 Trabalhos Futuros

Ao atingir aos objetivos propostos, pode-se dar seguimento no trabalho aplicando o *pipeline* proposto em outros projetos de comunidades *Open Source* e/ou aplicações comerciais, relatando e discutindo os resultados obtidos.

Outra oportunidade de trabalho futuro é construir outros *pipelines* DevSecOps, baseados neste apresentado nessa monografia, adicionando e/ou trocando as ferramentas *Open Source* empregadas no *pipeline* e as ferramentas de infraestrutura para comparação de resultados.

Acrescentar testes de integração orientando-se por abordagens de testes consolidadas é também um possível trabalho futuro, tendo em vista que a segurança de um sistema está correlacionada com o comportamento correto dos métodos e das funcionalidades.

Algo muito pertinente de ser mencionado nesse contexto de melhorias ao trabalho realizado, é o fato das constantes evoluções em termos de Segurança. Isso demanda constantes adequações no *pipeline* proposto, sendo muito interessante a colaboração de terceiros, para manter atualizado os insumos produzidos até o momento pelos autores.

Nesse sentido, os repositórios estão sob a Licença GNU, o que significa que são abertos a colaborações, desde que estejam em conformidade com o Guia de Contribuição.

## 7.5 Considerações Finais do Capítulo

Este capítulo detalhou as Questões de Pesquisa e Desenvolvimento, respondendo-as; e os Objetivos Geral e Específicos alcançados, com comprobatórios. Além disso, acordou-se sobre novas possibilidades de pesquisa a partir desse trabalho, destacadas em Trabalhos Futuros, com menção à necessidade de contínuas melhorias ao *pipeline* de DevSecOps, sob licença GNU, e incentivos às colaborações.

# Referências

ACTIONS, G. *Documentação do GitHub Actions*. 2023. Disponível em: <https://ghdocs-prod.azurewebsites.net/pt/actions>. Citado 3 vezes nas páginas 59, 60 e 61.

AKBAR, M. A. et al. Toward successful DevSecOps in software development organizations: A decision-making framework. *Information and Software Technology*, v. 147, p. 106894, jul. 2022. ISSN 09505849. Disponível em: <https://linkinghub.elsevier.com/retrieve/pii/S0950584922000568>. Citado 2 vezes nas páginas 53 e 54.

AKUJOBI, J. C. *A Model For Measuring Improvement Of Security In Continuous Integration pipelines : Metrics and Four-Axis Maturity Driven DevSecOps (MFAM) - University of Twente Student Theses*. 2021. Disponível em: <https://essay.utwente.nl/88916/>. Citado 2 vezes nas páginas 77 e 78.

ALBUQUERQUE, D. et al. *Detecção Interativa de Anomalias de Código-Um Estudo Experimental (final)*. [S.l.: s.n.], 2014. Citado na página 73.

ANDERSON, D. J. *Kanban: Successful Evolutionary Change for Your Technology Business*. [S.l.]: Blue Hole Press, 2010. ISBN 978-0-9845214-0-1. Citado na página 74.

BASS, L.; WEBER, I.; ZHU, L. *DevOps: a software architect's perspective*. New York Boston Indianapolis San Francisco Toronto Montreal London Munich Paris Madrid Capetown Sydney Tokyo Singapore Mexico City: Addison-Wesley, 2015. (The SEI series in software engineering). ISBN 978-0-13-404984-7 978-0-13-404988-5. Citado 3 vezes nas páginas 46, 48 e 72.

BECK, K. Embracing change with extreme programming. *Computer*, v. 32, n. 10, p. 70–77, out. 1999. ISSN 00189162. Disponível em: <http://ieeexplore.ieee.org/document/796139/>. Citado na página 75.

BECK, K. et al. Manifesto for agile software development. 2001. Publisher: Snowbird, UT. Disponível em: <https://ai-learn.it/wp-content/uploads/2019/03/03_ManifestoofAgileSoftwareDevelopment-1.pdf>. Citado na página 25.

BJöRNHOLM, J. *Performance of DevOps compared to DevSecOps : DevSecOps pipelines benchmarked!* [s.n.], 2020. Disponível em: <https://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-166985>. Citado 3 vezes nas páginas 35, 51 e 55.

BOOTSTRAP JACOB THORNTON, a. B. M. O. *Bootstrap*. 2024. Disponível em: <https://getbootstrap.com/>. Citado na página 59.

BOURQUE, P.; FAIRLEY, R. E. (Ed.). *SWEBOK: guide to the software engineering body of knowledge*. Version 3.0. Los Alamitos, CA: IEEE Computer Society, 2014. OCLC: 880350861. ISBN 978-0-7695-5166-1. Citado na página 26.

BOWEN, P.; HASH, J.; WILSON, M. Information security handbook: a guide for managers. 2006. Publisher: Pauline Bowen, Joan Hash, Mark Wilson. Disponível em: <https://www.nist.gov/publications/information-security-handbook-guide-managers?pub_id=50901>. Citado na página 26.

CERT.BR. *CERT.br - Estatísticas*. 2023. Disponível em: <https://stats.cert.br/>. Citado 2 vezes nas páginas 31 e 32.

COCKBURN, A. *Agile software development*. 4. print. ed. Boston, Mass. Munich: Addison-Wesley, 2003. (The Agile software development series). ISBN 978-0-201-69969-2. Citado na página 25.

CODEQL. *CódigoQL*. 2024. Disponível em: <https://codeql.github.com/>. Citado na página 64.

DEPENDABOT. *Guia de início rápido do Dependabot*. 2024. Disponível em: <https://docs.github.com/pt/code-security/getting-started/dependabot-quickstart-guide>. Citado na página 66.

Docker. *Why Docker | Docker*. 2021. Disponível em: <https://www.docker.com/why-docker/>. Citado na página 61.

DOCKER. *Home*. 2023. Disponível em: <https://docs.docker.com/>. Citado na página 62.

EBERT, C. et al. DevOps. *IEEE Software*, v. 33, n. 3, p. 94–100, maio 2016. ISSN 0740-7459, 1937-4194. Disponível em: <https://ieeexplore.ieee.org/document/7458761/>. Citado na página 26.

Express. *Express - framework de aplicativo da web Node.js*. 2023. Disponível em: <https://expressjs.com/pt-br/>. Citado na página 58.

F5. *F5.com*. 2023. Disponível em: <https://www.f5.com/>. Citado 10 vezes nas páginas 37, 38, 39, 40, 41, 42, 43, 44, 45 e 46.

FORSGREN, N.; HUMBLE, J. The Role of Continuous Delivery in it and Organizational Performance. *SSRN Electronic Journal*, 2015. ISSN 1556-5068. Disponível em: <http://www.ssrn.com/abstract=2681909>. Citado na página 49.

GERHARDT, T. E.; SILVEIRA, D. T. *Métodos de Pesquisa*. [S.l.]: Editora da UFRGS, 2009. ISBN 978-85-386-0071-8. Citado 2 vezes nas páginas 69 e 70.

GIL, A. C. *Como elaborar projetos de pesquisa*. [S.l.]: Editora Atlas Ltda, 2017. ISBN 978-85-970129-2-7. Citado 4 vezes nas páginas 70, 71, 74 e 78.

Git. *Git*. 2023. Disponível em: <https://git-scm.com/about>. Citado na página 57.

GITGUARDIAN. *GitGuardian: Git Security Scanning & Secrets Detection*. 2024. Disponível em: <https://www.gitguardian.com/>. Citado na página 66.

GitHub. *GitHub*. 2023. Disponível em: <https://github.com>. Citado na página 57.

GitHub Actions. *GitHub Actions*. 2023. Disponível em: <https://docs.github.com/en/actions>. Citado na página 59.

GOERTZEL, K. et al. *Software Security Assurance: A State-of-Art Report (SAR)*. [S.l.], 2007. Section: Technical Reports. Disponível em: <https://apps.dtic.mil/sti/citations/ADA472363>. Citado 2 vezes nas páginas 32 e 50.

GOLDRATT, E. M. *A Meta: Um Processo De Melhoria Continua*. [S.l.]: Nobel, 2002. ISBN 978-85-213-1236-9. Citado na página 46.

GUTTMAN, B.; ROBACK, E. An Introduction to Computer Security: the NIST Handbook. *NIST*, out. 1995. Last Modified: 2018-11-10T10:11-05:00 Publisher: Barbara Guttman, E Roback. Disponível em: <https://www.nist.gov/publications/introduction-computer-security-nist-handbook>. Citado na página 32.

GóMEZ, J. M. et al. (Ed.). *Engineering and Management of Data Centers: An IT Service Management Approach*. Cham: Springer International Publishing, 2017. (Service Science: Research and Innovations in the Service Economy). ISBN 978-3-319-65081-4 978-3-319-65082-1. Disponível em: <http://link.springer.com/10.1007/978-3-319-65082-1>. Citado na página 47.

HERKOU. *Documentation | Heroku Dev Center*. 2023. Disponível em: <https://devcenter.heroku.com/categories/reference>. Citado na página 61.

HEROKU. *Cloud Application Platform | Heroku*. 2023. Disponível em: <https://www.heroku.com/>. Citado na página 61.

HUSEBY, S. H. *Innocent Code: A Security Wake-Up Call for Web Programmers*. 1st edition. ed. Chichester: Wiley, 2004. ISBN 978-0-470-85744-1. Citado na página 35.

Internet World Stats. *Internet World Stats*. 2023. Disponível em: <https://www.internetworldstats.com/>. Citado na página 85.

JUNIOR, J. R. d. A. *SEGURANÇA EM SISTEMAS CRÍTICOS E EM SISTEMAS DE INFORMAÇÃO – UM ESTUDO COMPARATIVO*. Tese (Doutorado) — Escola Politécnica da Universidade de São Paulo, 2003. Citado na página 102.

KIM, G.; BEHR, K.; SPAFFORD, G. *The phoenix project: a novel about IT, DevOps, and helping your business win*. Third edition. Portland, OR: IT Revolution, 2018. OCLC: ocn993043051. ISBN 978-1-942788-29-4. Citado 3 vezes nas páginas 46, 47 e 72.

KIM, G. et al. *The DevOps handbook: how to create world-class agility, reliability, & security in technology organizations*. First edition. Portland, OR: IT Revolution Press, LLC, 2016. ISBN 978-1-942788-00-3. Citado 2 vezes nas páginas 49 e 72.

KIM, G. et al. *The DevOps Handbook: How to Create World-Class Agility, Reliability, & Security in Technology Organizations*. [S.l.]: IT Revolution, 2021. Google-Books-ID: 8kRDEAAAQBAJ. ISBN 978-1-950508-43-3. Citado 2 vezes nas páginas 26 e 46.

LAN, F.; ZHANG, J. Database Security——SQL Injection. 2017. Citado na página 39.

LAUKKANEN, E.; ITKONEN, J.; LASSENIUS, C. Problems, causes and solutions when adopting continuous delivery—A systematic literature review. *Information and Software Technology*, v. 82, p. 55–79, fev. 2017. ISSN 09505849. Disponível em: <https://linkinghub.elsevier.com/retrieve/pii/S0950584916302324>. Citado na página 49.

Lean Enterprise Institute. *What is Lean? | Lean Thinking*. 2023. Disponível em: <https://www.lean.org/explore-lean/what-is-lean/>. Citado na página 46.

LIKERT, R. A Technique for the Measurement of Attitudes. *Archives of Psychology*, v. 140, p. 1–55, 1932. Citado na página 70.

Martin Fowler. *Continuous Delivery*. 2013. Citado na página 50.

MCGRAW, G. Software security. *IEEE Security & Privacy*, v. 2, n. 2, p. 80–83, mar. 2004. ISSN 1558-4046. Conference Name: IEEE Security & Privacy. Citado 2 vezes nas páginas 31 e 33.

Microsoft. *Microsoft Security Development Lifecycle (SDL) Process Guidance - Version 5.2*. 2016. Disponível em: <https://www.microsoft.com/en-us/download/details.aspx?id=29884>. Citado na página 34.

MUI. *MUI*. 2023. Disponível em: <https://mui.com/about/>. Citado na página 59.

MUñOZ, M.; NEGRETE, M.; MEJíA, J. Proposal to Avoid Issues in the DevOps Implementation: A Systematic Literature Review. In: ROCHA, et al. (Ed.). *New Knowledge in Information Systems and Technologies*. Cham: Springer International Publishing, 2019. v. 930, p. 666–677. ISBN 978-3-030-16180-4 978-3-030-16181-1. Series Title: Advances in Intelligent Systems and Computing. Disponível em: <http://link.springer.com/10.1007/978-3-030-16181-1_63>. Citado na página 48.

MYRBAKKEN, H.; COLOMO-PALACIOS, R. DevSecOps: A Multivocal Literature Review. In: MAS, A. et al. (Ed.). *Software Process Improvement and Capability Determination*. Cham: Springer International Publishing, 2017. (Communications in Computer and Information Science), p. 17–29. ISBN 978-3-319-67383-7. Citado 2 vezes nas páginas 26 e 27.

Node. *Node.js*. 2023. Disponível em: <https://nodejs.org/en>. Citado na página 58.

npm. *About npm*. 2023. Disponível em: <https://docs.npmjs.com/about-npm>. Citado na página 58.

OUELLET, M. et al. The Network of Online Stolen Data Markets: How Vendor Flows Connect Digital Marketplaces. *The British Journal of Criminology*, v. 62, n. 6, p. 1518–1536, nov. 2022. ISSN 0007-0955. Disponível em: <https://doi.org/10.1093/bjc/azab116>. Citado na página 39.

OWASP Dependency Check. *OWASP Dependency-Check | OWASP Foundation*. 2023. Disponível em: <https://owasp.org/www-project-dependency-check/>. Citado na página 65.

OWASP Foundation. *About the OWASP Foundation*. 2001. Disponível em: <https://owasp.org/about/>. Citado na página 35.

OWASP Foundation. *OWASP Foundation, the Open Source Foundation for Application Security*. 2001. Disponível em: <https://owasp.org/>. Citado 2 vezes nas páginas 26 e 27.

OWASP Foundation. *OWASP Top Ten*. 2003. Disponível em: <https://owasp.org/www-project-top-ten/>. Citado 3 vezes nas páginas 35, 36 e 72.

OWASP Foundation. *A01 Broken Access Control - OWASP Top 10:2021*. 2021. Disponível em: <https://owasp.org/Top10/A01_2021-Broken_Access_Control/>. Citado na página 37.

OWASP Foundation. *A02 Cryptographic Failures - OWASP Top 10:2021*. 2021. Disponível em: <https://owasp.org/Top10/A02_2021-Cryptographic_Failures/>. Citado 2 vezes nas páginas 37 e 38.

OWASP Foundation. *A03 Injection - OWASP Top 10:2021*. 2021. Disponível em: <https://owasp.org/Top10/A03_2021-Injection/>. Citado na página 39.

OWASP Foundation. *A04 Insecure Design - OWASP Top 10:2021*. 2021. Disponível em: <https://owasp.org/Top10/A04_2021-Insecure_Design/>. Citado na página 39.

OWASP Foundation. *A05 Security Misconfiguration - OWASP Top 10:2021*. 2021. Disponível em: <https://owasp.org/Top10/A05_2021-Security_Misconfiguration/>. Citado na página 40.

OWASP Foundation. *A06 Vulnerable and Outdated Components - OWASP Top 10:2021*. 2021. Disponível em: <https://owasp.org/Top10/A06_2021-Vulnerable_and_Outdated_Components/>. Citado na página 41.

OWASP Foundation. *A07 Identification and Authentication Failures - OWASP Top 10:2021*. 2021. Disponível em: <https://owasp.org/Top10/A07_2021-Identification_and_Authentication_Failures/>. Citado na página 42.

OWASP Foundation. *A08 Software and Data Integrity Failures - OWASP Top 10:2021*. 2021. Disponível em: <https://owasp.org/Top10/A08_2021-Software_and_Data_Integrity_Failures/>. Citado na página 43.

OWASP Foundation. *A09 Security Logging and Monitoring Failures - OWASP Top 10:2021*. 2021. Disponível em: <https://owasp.org/Top10/A09_2021-Security_Logging_and_Monitoring_Failures/>. Citado na página 44.

OWASP Foundation. *A10 Server Side Request Forgery (SSRF) - OWASP Top 10:2021*. 2021. Disponível em: <https://owasp.org/Top10/A10_2021-Server-Side_Request_Forgery_%28SSRF%29/>. Citado na página 45.

OWASP Foundation. *OWASP Dependency-Check*. 2023. Disponível em: <https://owasp.org/www-project-dependency-check/>. Citado na página 51.

OWASP Foundation. *SQL Injection Prevention - OWASP Cheat Sheet Series*. 2023. Disponível em: <https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html#primary-defenses>. Citado na página 39.

PRASANNA, K. et al. PoC Design: A Methodology for Proof-of-Concept (PoC) Development on Internet of Things Connected Dynamic Environments. *Security and Communication Networks*, v. 2021, p. 1–12, out. 2021. ISSN 1939-0122, 1939-0114. Disponível em: <https://www.hindawi.com/journals/scn/2021/7185827/>. Citado na página 72.

PRATES, L. et al. DevSecOps Metrics. In: WRYCZA, S.; MAśLANKOWSKI, J. (Ed.). *Information Systems: Research, Development, Applications, Education*. Cham: Springer International Publishing, 2019. (Lecture Notes in Business Information Processing), p. 77–90. ISBN 978-3-030-29608-7. Citado 2 vezes nas páginas 77 e 147.

RANSOME, J.; MISRA, A. *Core Software Security*. 0. ed. Auerbach Publications, 2018. ISBN 978-1-4665-6096-3. Disponível em: <https://www.taylorfrancis.com/books/9781466560963>. Citado 2 vezes nas páginas 31 e 34.

React. *React*. 2023. Disponível em: <https://react.dev/>. Citado na página 58.

RIES, E. Minimum viable product: a guide. *Startup lessons learned*, v. 3, n. 1, 2009. Disponível em: <http://soloway.pbworks.com/w/file/fetch/85897603/1%2B%20Les-sons%20Learned_%20Minimum%20Viable%20Product_%20a%20guide2.pdf>. Citado na página 25.

SCHWABER, K.; SUTHERLAND, J. *Scrum Guide | Scrum Guides*. 2020. Disponível em: <https://scrumguides.org/scrum-guide.html>. Citado na página 74.

ScienceDirect. *Denial-of-Service - an overview | ScienceDirect Topics*. 2023. Disponível em: <https://www.sciencedirect.com/topics/computer-science/denial-of-service>. Citado na página 46.

ScienceDirect. *Remote Code Execution - an overview | ScienceDirect Topics*. 2023. Disponível em: <https://www.sciencedirect.com/topics/computer-science/remote-code-execution>. Citado na página 45.

SILVA, K. S. L. d.; ALVES, D. B. Cenários Investigativos no Software RStudio para Educação Matemática. p. 6, 2018. Disponível em: <https://www.uft.edu.br/matematicaaraguaina/includes/eventos/2018/Comunicacao/CC5.pdf>. Citado 2 vezes nas páginas 29 e 158.

SNYK. *Getting started | Snyk User Docs*. 2024. Disponível em: <https://docs.snyk.io/getting-started>. Citado na página 66.

SONARQUBE. *SonarQube 10.3*. 2023. Disponível em: <https://docs.sonarsource.com/sonarqube/latest/>. Citado 2 vezes nas páginas 63 e 64.

SonarQube. *Try Now Enterprise Edition*. 2023. Disponível em: <https://www.sonarsource.com/products/sonarqube/enterprise-edition/>. Citado na página 62.

SQLMAP. *sqlmap: automatic SQL injection and database takeover tool*. 2023. Disponível em: <https://sqlmap.org/>. Citado na página 67.

STALLINGS, W. *Cryptography and network security: principles and practice*. Seventh edition, global edition. Boston Columbus Indianapolis New York San Francisco Hoboken Amsterdam Cape Town Dubai London Madrid Milan Munich Paris Montreal Toronto Delhi Mexico City Sao Paulo Sydney Hong Kong Seoul Singapore Taipei Tokyo: Pearson, 2017. ISBN 978-1-292-15858-7. Citado 4 vezes nas páginas 26, 32, 34 e 72.

sullo. *nikto*. 2023. Original-date: 2012-11-24T04:24:29Z. Disponível em: <https://github.com/sullo/nikto>. Citado na página 68.

SUTHERLAND, J. Business object design and implementation workshop. In: *Addendum to the proceedings of the 10th annual conference on Object-oriented programming systems, languages, and applications*. Austin Texas USA: ACM, 1995. p. 170–175. ISBN 978-0-89791-721-6. Disponível em: <https://dl.acm.org/doi/10.1145/260094.260274>. Citado na página 74.

TECHNOLOGIES, S. *AGILE METHODOLOGIES Survey Results*. 2010. Disponível em: <https://web.archive.org/web/20100821225423/http://www.shinetech.com/attachments/104_ShineTechAgileSurvey2003-01-17.pdf>. Citado na página 25.

UNIVERSAL, A. *World Security Report 2023*. [S.l.], 2023. 55 p. Disponível em: <https://www.worldsecurityreport.com/media/v1ahrj2v/a4_world-security-report_vf_en.pdf>. Citado na página 27.

VALENTE, M. T. *Engenharia de Software Moderna: Princípios e Práticas para Desenvolvimento de Software com Produtividade*. [S.l.]: Editora: Independente, 2020. Citado na página 35.

VEHENT, J. *Securing DevOps: security in the Cloud*. Shelter Island, New York: Manning Publications Co, 2018. OCLC: on1050870710. ISBN 978-1-61729-413-6. Citado 5 vezes nas páginas 26, 49, 50, 51 e 72.

VERCEL. *Get started with Vercel*. 2024. Disponível em: <https://vercel.com/docs/getting-started-with-vercel>. Citado na página 61.

VIEGA, J.; MCGRAW, G. *Building secure software: how to avoid security problems the right way*. 1. print. ed. Boston, Mass. Munich: Addison-Wesley, 2006. (Software security library, buil). ISBN 978-0-321-42523-2. Citado na página 31.

Visual Studio Code. *Visual Studio Code*. 2023. Disponível em: <https://code.visualstudio.com/>. Citado na página 57.

WOTAWA, F. On the Automation of Security Testing. In: *2016 International Conference on Software Security and Assurance (ICSSA)*. [s.n.], 2016. p. 11–16. Disponível em: <https://ieeexplore.ieee.org/document/7861644>. Citado na página 34.

ZAP. *ZAP – Documentation*. 2023. Disponível em: <https://www.zaproxy.org/docs/>. Citado na página 66.

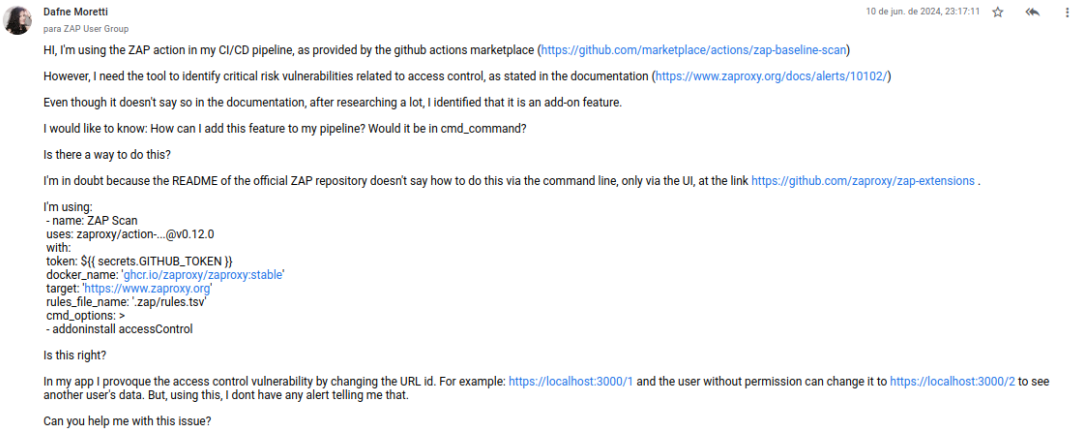
Zotero. *Zotero*. 2023. Disponível em: <https://www.zotero.org/about/>. Citado na página 72.

# Apêndices

# APÊNDICE A – Debate sobre identificação de controle de acesso na ferramenta ZAP

Os autores contataram o mantenedor da ferramenta Owasp ZAP para compreender melhor como poderiam acrescentar no *pipeline* CICD o controle de acesso na ferramenta, disponível em seu ADD-ON. Este primeiro contato foi apresentado na Figura 91.

Figura 91 – A01 Controle de Acesso OWASP ZAP



**Dafne Moretti** 10 de jun. de 2024, 23:17:11

para ZAP User Group

HI, I'm using the ZAP action in my CI/CD pipeline, as provided by the github actions marketplace (https://github.com/marketplace/actions/zap-baseline-scan)

However, I need the tool to identify critical risk vulnerabilities related to access control, as stated in the documentation (https://www.zaproxy.org/docs/alerts/10102/)

Even though it doesn't say so in the documentation, after researching a lot, I identified that it is an add-on feature.

I would like to know: How can I add this feature to my pipeline? Would it be in cmd_command?

Is there a way to do this?

I'm in doubt because the README of the official ZAP repository doesn't say how to do this via the command line, only via the UI, at the link https://github.com/zaproxy/zap-extensions .

I'm using:
- name: ZAP Scan
uses: zaproxy/action-...@v0.12.0
with:
token: ${{ secrets.GITHUB_TOKEN }}
docker_name: 'ghcr.io/zaproxy/zaproxy:stable'
target: 'https://www.zaproxy.org'
rules_file_name: '.zap/rules.tsv'
cmd_options: >
- addoninstall accessControl

Is this right?

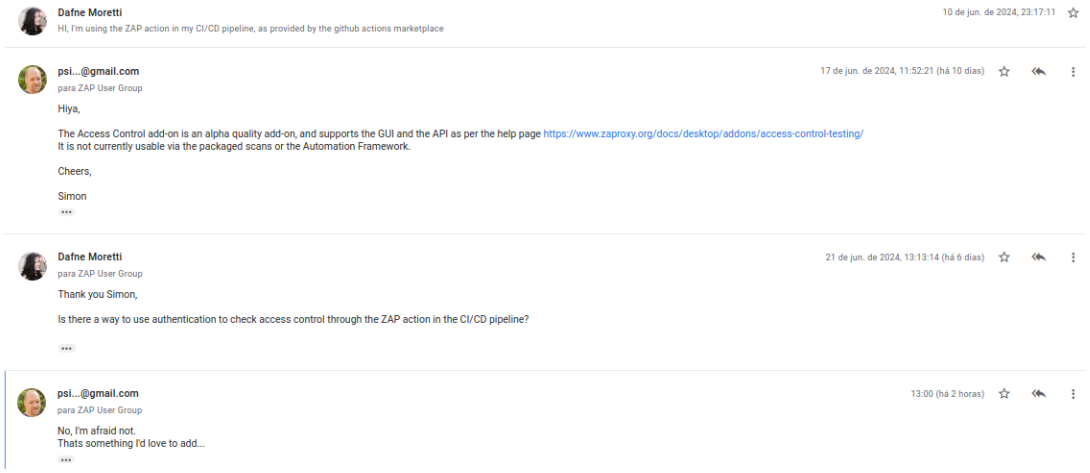
In my app I provoque the access control vulnerability by changing the URL id. For example: https://localhost:3000/1 and the user without permission can change it to https://localhost:3000/2 to see another user's data. But, using this, I dont have any alert telling me that.

Can you help me with this issue?

Fonte: Autores

Para isso, os autores entraram no grupo para discussões da ferramenta no Gmail e explicaram a situação que estavam enfrentando. A resposta para o problema levantado pode ser vista na Figura 92.

Figura 92 – A01 Controle de Acesso OWASP ZAP Resposta



**Dafne Moretti** 10 de jun. de 2024, 23:17:11

HI, I'm using the ZAP action in my CI/CD pipeline, as provided by the github actions marketplace

**psi...@gmail.com** 17 de jun. de 2024, 11:52:21 (há 10 dias)

para ZAP User Group

Hiya,

The Access Control add-on is an alpha quality add-on, and supports the GUI and the API as per the help page https://www.zaproxy.org/docs/desktop/addons/access-control-testing/
It is not currently usable via the packaged scans or the Automation Framework.

Cheers,

Simon

**Dafne Moretti** 21 de jun. de 2024, 13:13:14 (há 6 dias)

para ZAP User Group

Thank you Simon,

Is there a way to use authentication to check access control through the ZAP action in the CI/CD pipeline?

**psi...@gmail.com** 13:00 (há 2 horas)

para ZAP User Group

No, I'm afraid not.
Thats something I'd love to add...

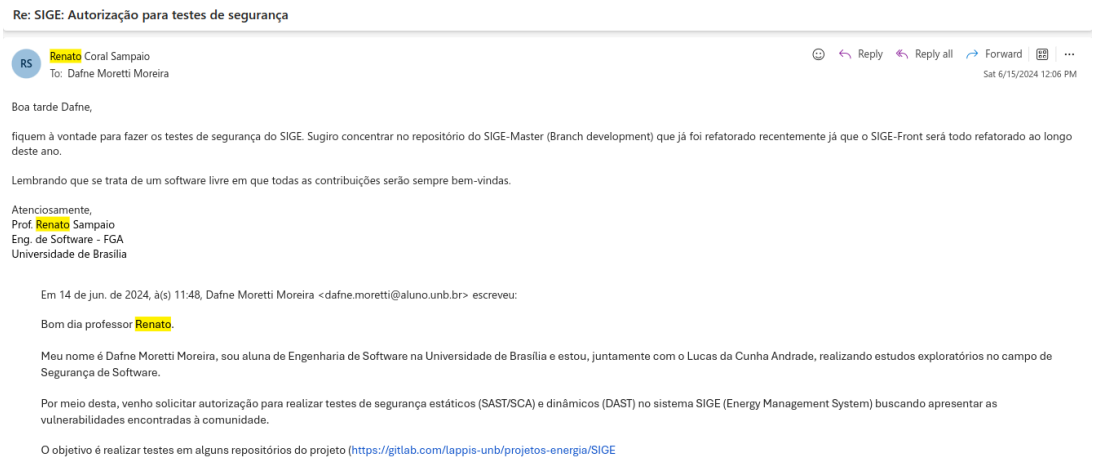
Fonte: Autores

Não há como adicionar controle de acesso no CI/CD até o presente momento de elaboração da monografia.

# APÊNDICE B – Autorização para testes de segurança no SIGE

Para realizar os testes no SIGE, os alunos solicitaram autorização para o mantenedor (vide Figura 93).

Figura 93 – Email de autorização de testes no SIGE



**Re: SIGE: Autorização para testes de segurança**

RS Renato Coral Sampaio
To: Dafne Moretti Moreira

Reply | Reply all | Forward | ...
Sat 6/15/2024 12:06 PM

Boa tarde Dafne,

fiquem à vontade para fazer os testes de segurança do SIGE. Sugiro concentrar no repositório do SIGE-Master (Branch development) que já foi refatorado recentemente já que o SIGE-Front será todo refatorado ao longo deste ano.

Lembrando que se trata de um software livre em que todas as contribuições serão sempre bem-vindas.

Atenciosamente,
Prof. Renato Sampaio
Eng. de Software - FGA
Universidade de Brasília

Em 14 de jun. de 2024, à(s) 11:48, Dafne Moretti Moreira <dafne.moretti@aluno.unb.br> escreveu:

Bom dia professor Renato.

Meu nome é Dafne Moretti Moreira, sou aluna de Engenharia de Software na Universidade de Brasília e estou, juntamente com o Lucas da Cunha Andrade, realizando estudos exploratórios no campo de Segurança de Software.

Por meio desta, venho solicitar autorização para realizar testes de segurança estáticos (SAST/SCA) e dinâmicos (DAST) no sistema SIGE (Energy Management System) buscando apresentar as vulnerabilidades encontradas à comunidade.

O objetivo é realizar testes em alguns repositórios do projeto (https://gitlab.com/lappis-unb/projetos-energia/SIGE

Fonte: Autores