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

Segurança de Software Open Source: Impacto das Melhores Práticas nos Indicadores de Vulnerabilidade do Código

Autor: João Pedro Alves da Silva Chaves
Orientadora: Dra. Elaine Venson

Brasília, DF
2023

João Pedro Alves da Silva Chaves

Segurança de Software Open Source: Impacto das Melhores Práticas nos Indicadores de Vulnerabilidade do Código

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: Dra. Elaine Venson

Brasília, DF

2023

Resumo

O Movimento de Programa de Código Aberto tem crescido nas últimas três décadas, inclusive com aumento da utilização por parte de organizações em 2022. No entanto, esse crescente uso também tem resultado em mais vulnerabilidades que podem levar a violações de direitos e ataques cibernéticos. A Engenharia de Software busca reduzir essas vulnerabilidades e aumentar a segurança, com modelos de boas práticas de desenvolvimento de software seguro como SDL, SAMM e BSIMM para software em geral. O Programa de Selo de Melhores Práticas da OpenSSF (PSMPO) foi desenvolvido para auxiliar o desenvolvimento e manutenção de projetos de código aberto incentivando a implementação de melhores práticas de segurança. Este trabalho tem como objetivo identificar se há uma relação entre o emprego de práticas do processo PSMPO no desenvolvimento de softwares de código aberto e seu nível de segurança indicado pelas densidades de vulnerabilidades, de code smells e de hotspots de segurança. Mais especificamente, foram identificadas as práticas aplicadas em cada projeto por meio de uma pesquisa telematizada em plataforma do PSMPO, as quais foram relacionadas com as densidades de vulnerabilidades, de code smells e de hotspots de segurança obtidas por meio do teste estático de segurança nesses softwares. Como resultado, foi averiguado pela aplicacao da analise exploratoria de dados que o incentivo dado pelo PSMPO para a adoção de práticas de segurança tem baixo efeito na redução das vulnerabilidades dos projetos de código aberto e que mais de 60% dos projetos nao aplica nenhuma prática de segurança. Em futuros trabalhos, é possível realizar uma comparação entre as densidades de vulnerabilidades, de code smells e de hotspots de segurança em projetos que adotam as melhores práticas do PSMPO com aqueles que não seguem tais práticas em programas de código aberto.

Palavras-chave: OSS; Vulnerabilidade; Segurança de Software.

Abstract

The Open Source Source Movement has experienced growth over the past three decades, with a notable increase in adoption by organizations in 2022. However, this heightened utilization has concurrently led to an escalation in vulnerabilities, posing risks of rights violations and cyber attacks. The field of Software Engineering is dedicated to mitigating these vulnerabilities and enhancing security, utilizing models of best practices for secure software development, such as SDL, SAMM, and BSIMM, applicable to software in a general context. The OpenSSF Best Practices Badge Program (OBPBP) has been developed to provide assistance in the development and maintenance of open source projects by fostering the implementation of security best practices. The objective of this study is to discern whether a correlation exists between the application of OBPBP practices in open source software development and the resultant security level, as indicated by densities of vulnerability, code smells and security hotspots. More specifically, the practices implemented in each project were identified through a telematized survey on the OBPBP platform, and these were subsequently correlated with vulnerability densities, code smells, and security hotspots identified through static security testing in these software projects. The outcomes, revealed through the application of exploratory data analysis, suggest that the encouragement provided by OBPBP to adopt security practices has a small impact on reducing vulnerabilities in open source projects, with over 60% of projects not implementing any security practices. In future endeavors, a comparative analysis may be conducted between densities of vulnerability, code smells and security hotspots in projects adhering to OBPBP best practices and those that deviate from such practices in open source software.

Key-words: latex. abntex. text editoration.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

API	Application Programming Interface
BSIMM	Building Security in Maturity Model
CWE	Common Weakness Enumeration
FLOSS	Free/Libre Open Source Software
OBPBP	OpenSSF Best Practices Badge Program
OpenSSF	Open Source Security Foundation
OSI	Open Source Initiative
OSS	Open Source Sofware
OWASP	Open Worldwide Application Security Project
PSMPO	Programa de Selo de Melhores Práticas da OpenSSF
SAMM	Software Assurance Maturity Model
SDL	Security Development Lifecycle
SSDL	Secure Software Development Lifecycle
SSG	Software Security Group
URL	Uniform Resource Locator

Sumário

1 Introdução

1.1 Considerações Iniciais do Capítulo

Este capítulo inicial aborda o contexto no qual este trabalho está inserido, o problema de pesquisa, o objetivo geral e específicos, a metodologia de pesquisa empregada e a organização dos capítulos subsequentes.

1.2 Contexto

O Movimento de Programa de Código Aberto (*Open Source Software* - OSS) tem se desenvolvido de forma crescente, principalmente nas últimas três décadas. Iniciativas de apoio e requisitos tem sido propostas pela organização *Iniciativa de Código Aberto* (*Open Source Iniciative* - OSI) (Open Source Iniciative, 2006), as quais norteiam a produção e contribuição dos códigos abertos pela comunidade.

O termo *Open Source Software* foi proposto pela *Christine Peterson* do *Foresight Institute* para se desvincular do termo "*free software*" (BRETTHAUER, 2001), ressaltando o sentido de liberdade. Embora em português, o termo "*free software*" possa ser traduzido como programa gratuito, o adequado é traduzi-lo como "programa livre"(Free Software Foundation, 2019).

O *Movimento de Programa de Código Aberto* tem se tornado cada vez mais forte e organizado, com altas taxas de adesão e de interesse, conforme apresentado no Relatório do Estado do Código Aberto 2023 (*2023 State of Open Source Report*) produzido pela *Open Logic* e pela *Open Source Initiative* (OSI). O relatório apresenta um aumento de mais de 80% das organizações participantes no ano de 2022 utilizando programas de código aberto (OpenLogic by Perforce, 2023).

Com o aumento da utilização de programas de código aberto, os casos de vulnerabilidades de código aberto se tornam mais presentes, conforme pesquisa sobre segurança em código aberto e análise de risco do Centro de Pesquisa de Cybersegurança (*Cybersecurity Research Center* - CyRC) da empresa Synopsys. A pesquisa aponta que, em 2022, 84% das bases de código continham pelo menos uma vulnerabilidade de código aberto (Synopsys, Inc., 2023).

A presença de vulnerabilidades em um software pode possibilitar ataques cibernéticos, e esses, resultarem em violações de direitos dos usuários, funcionários ou até mesmo da própria empresa. As vulnerabilidades com maior impacto geralmente são encontradas em programas de código aberto ou "programas livres". Nesse contexto, a *segurança de*

software é essencial (SáNCHEZ et al., 2020).

Pesquisas e desenvolvimentos da Engenharia de Software tem buscado a redução de vulnerabilidades e o aumento da segurança. Alguns modelos de boas práticas em desenvolvimento de software seguro foram propostos, como por exemplo: o *Security Development Lifecycle* (SDL), *Software Assurance Maturity Model* (SAMM) e *Building Security in Maturity Model* (BSIMM). Em cada modelo, estão inseridas práticas que podem estar diretamente ligadas à segurança no processo ou às melhores práticas de desenvolvimento seguro do software. Em alguns casos, as práticas dos processos (SDL, SAMM, e BSIMM) se repetem, contudo nomeadas diferentemente ou com pequenas alterações na estrutura.

Com o intuito de auxiliar os projetos de código aberto no aumento de qualidade e segurança pela aplicação das melhores práticas foi desenvolvido pela *Open Source Software Foundation* um sistema de autoavaliação das práticas dos projetos, chamado *Programa de Selos de Melhores Práticas da OpenSSF* (PSMPO), do inglês *OpenSSF Best Practices Badge Program*[1].

O PSMPO apresenta uma série de dados sobre os projetos de Código Aberto que passaram pela autoavaliação, incluindo quais práticas são aplicadas por cada um deles (Open Source Security Foundation, 2021a). Uma das práticas identificáveis é a análise estática do código, a qual pertence ao conjunto de práticas voltadas para análise.

1.3 Problema

Com a proposição e emprego de práticas para a redução da vulnerabilidade e o aumento da segurança como (*Security Development Lifecycle* (SDL), *Software Assurance Maturity Model* (SAMM) e *Building Security in Maturity Model* (BSIMM)), inclusive para Código Aberto (Programa de Selo de Melhores Práticas da OpenSSF), novas oportunidades de contribuição para a área são abertas.

No caso do PSMPO (Programa de Selo de Melhores Práticas da OpenSSF), uma lacuna é a falta de estudos sobre o impacto do emprego das práticas de desenvolvimento de software do programa nos indicadores de densidade de vulnerabilidades, de "*code smells*" e de "*hotspots*" de segurança desse software.

Nesse contexto, a pergunta de pesquisa é: ***há uma relação entre o emprego de práticas do programa PSMPO no desenvolvimento de OSS (Open Source Software) e as densidades de vulnerabilidades, de code smells e de hotspots de segurança dos OSS****?*

[1] https://bestpractices.coreinfrastructure.org/pt-BR

1.4 Objetivo

1.4.1 Objetivo Geral

O objetivo geral deste trabalho é *identificar se há uma relação entre o emprego de práticas do processo PSMPO no desenvolvimento de softwares de código aberto e seu nível de segurança indicado pelas densidades de vulnerabilidades, de code smells e de hotspots de segurança.*

1.4.2 Objetivos Específicos

Os objetivos específicos deste trabalho são:

1. Identificar a densidade de vulnerabilidades, a densidade de *code smells* e a densidade de *hotspots* de segurança dos OSSs selecionados da base PSMPO.
2. Identificar as práticas do PSMPO empregadas em cada um dos OSSs selecionados da base PSMPO.
3. Aplicar análise exploratória de dados para calcular coeficiente de correlação entre a densidade de vulnerabilidades, a densidade de *code smells* e a densidade de *hotspots* de segurança em relação ao número de práticas do PSMPO.

1.5 Metodologia

Quanto ao objetivo, esta pesquisa é classificada como descritiva (GIL, 2002), pois por meio dos dados coletados e analisados descreve a relação do emprego do PSMPO e as densidades de vulnerabilidades, de *code smells* e de *hotspots* de segurança de uma amostra de projetos de Código Aberto. Quanto à abordagem, é uma pesquisa quantitativa, pois utiliza métodos e técnicas estatísticas.

A pesquisa foi dividida em sete etapas, sendo uma adaptação da diagramação geral de pesquisa (GIL, 2002), com a retirada de algumas etapas que não seriam necessárias nesta pesquisa. As sete etapas: *formulação do problema*, *operacionalização das variáveis*, *elaboração dos instrumentos de coleta de dados*, *seleção da amostra*, *coleta de dados*, *análise e interpretação dos dados*, e *relatório dos resultados* etapas estão detalhadas no Capítulo Metodologia na seção Planejamento da Pesquisa.

São utilizados *scripts* de *web scraping* para coletar e organizar os dados dos sites do PSMPO e do *SonarCloud*, plataforma de análise estática. Os dados de interesse são quantas e quais as práticas do PSMPO são empregadas por OSSs que estão listados na base de projetos do próprio PSMPO. Outros dados de interesse são o número de linhas

de código e as quantidades de vulnerabilidades, de *code smells* e de *hotspots* de segurança identificadas por ferramenta de análise estática.

Após a coleta de dados, na etapa de análise são utilizadas técnicas estáticas como cálculo de coeficiente de correlação e apresentação de gráficos de dispersão. Estas técnicas são aplicadas para verificar se há uma correlação entre as densidades de vulnerabilidades, de *code smells* e de *hotspots* de segurança e a quantidade de práticas empregadas do PSMPO.

1.6 Organização do Trabalho

Este trabalho possui seis capítulos. Neste Capítulo 1 - *Introdução*, foram apresentados: contexto do trabalho, o problema a ser abordado, os objetivos e a metodologia de pesquisa a ser utilizada.

No Capítulo 2 - *Segurança de Software* são apresentados conceitos relacionados à segurança de software, desenvolvimento seguro de software e três processos que incluem a segurança desde a etapa de design da aplicação: *Security Development Lifecycle* e SAMM.

No Capítulo 3 - *Metodologia* é apresentado o planejamento da pesquisa. O planejamento está dividido em seis passos, os quais estão apresentados por sessões.

No Capítulo 4 - *Coleta de Dados* são apresentados os passos utilizados pelos *scripts* para coleta das métricas e dados dos projetos .

No Capítulo 5 - *Análise de Dados* são apresentados os cálculos dos coeficientes de correlação, os diagramas de caixa, os histogramas e os gráficos de dispersão utilizados para a análise estatística.

No Capítulo 6 - *Conclusão e Trabalhos Futuros* são apresentados os resultados dos objetivos específicos, a conclusão deste trabalho e os trabalhos futuros a serem desenvolvidos para o avanço no conhecimento sobre a relação entre o PSMPO e a segurança de software.

2 Segurança de Software

2.1 Considerações Iniciais do Capítulo

Neste capítulo, apresenta-se a Segurança de Software, com a abordagem de aspectos como o Desenvolvimento Seguro de Software e dois modelos, o Ciclo de Vida de Desenvolvimento Seguro e o SAMM. Além disso, são explorados conceitos como Software de Código Aberto e a estrutura das comunidades associadas. A conexão entre a segurança de software e o software de código aberto é destacada, inclusa a apresentação do Programa de Selo de Melhores Práticas da OpenSSF. Para complementar, uma explicação sobre a análise Exploratória de Dados é fornecida.

2.2 Segurança de Software

A segurança de software não pode ser definida de forma simplista (VIEGA; MCGRAW, 2002), porém pode ser compreendida como a ideia de projetar aplicações de maneira que funcionem corretamente mesmo em situações de ataques. Portanto, inclui os processos de projetar, construir e testar para segurança (MCGRAW, 2004). Além disso, por causa da ampla utilização de programas é importante que sejam seguros por *design* (RANSOME; MISRA, 2013), com o objetivo de minimizar ao máximo as ocorrências de vulnerabilidades.

Uma falha de segurança em um software é um exemplo de uma situação em que não foi alcançado o objetivo de segurança, e uma vulnerabilidade é a causa de tal falha (PIESSENS, 2021). O que demonstra que a segurança de um *software* possui três pilares: projeto para ser seguro, garantia de segurança e educação dos desenvolvedores, arquitetos e usuários sobre como construir coisas seguras (MCGRAW, 2004).

Estes três pontos são percebidos ao analisar relatórios como o "Top 10 vulnerabilidades" desenvolvido pelo Projeto de Segurança de Aplicativos Aberto em Nível Mundial (*Open Worldwide Application Security Project* - OWASP)(The OWASP Foundation, 2021) e também o relatório anual da Enumeração de Fraquezas Comuns (*Common Weakness Enumeration* - CWE) chamado *Top 25 Most Dangerous Software Weaknesses*(Common Weakness Enumeration, 2022), que em tradução livre são as 25 fraquezas de software mais perigosas.

Nestes relatórios é possível notar a presença de falhas e vulnerabilidades relacionadas à ausência de um projeto para a aplicação ser segura, com o exemplo da vulnerabilidade "design inseguro" (The OWASP Foundation, 2021). Outro exemplo é a fraqueza

de *"overflow"* de inteiro que é apresentada em décimo terceiro lugar no Top 25 fraquezas de *software* de 2022 de acordo com a CWE (Common Weakness Enumeration, 2022) e está relacionada à falta de conhecimento dos desenvolvedores sobre como desenvolver com segurança.

Outros conceitos relevantes para a área de segurança de software são *code smells* e *hotspots* de segurança. *Code smells* são trechos de código em que não são aplicados padrões de projeto de software (SINGH; KAUR, 2018), de modo em que há um prejuízo na manutenibilidade do código, por consequinte há maiores chances de inserção de vulnerabilidades. *Hotspots* são trechos de código em que foi identificada uma maior possibilidade de haver vulnerabilidades e portanto, é necessária uma verificação manual (SonarSource S.A, 2023b).

É relevante a compreensão de que a segurança de *software* é sobre o entendimento dos riscos de segurança causados por programas e como lidar com eles (MCGRAW, 2006). Até mesmo, porque a maior causa dos problemas de segurança é a falha ou mau funcionamento de uma aplicação de maneira inesperada (VIEGA; MCGRAW, 2002).

Com o objetivo de minimizar os riscos de *software* inseguro, diversas soluções de segurança de Tecnologia da Informação (TI) foram desenvolvidas. No entanto, para justificar a implementação dessas soluções, é crucial entender os riscos e custos financeiros associados à falta de integração da segurança à aplicação. Pois, a segurança de software é uma decisão de negócios e deve ser cuidadosamente pensada e analisada (RANSOME; MISRA, 2013).

Em 2004, as organizações começaram criar processos para inserção de segurança nos ciclos de vida de desenvolvimento de software (GEER, 2010), o que caracteriza o desenvolvimento seguro de software. As boas práticas de segurança de *software* ajudam a garantir o funcionamento adequado da aplicação (VIEGA; MCGRAW, 2002).

2.3 Desenvolvimento Seguro de Software

As melhores práticas de segurança de software unem boas práticas da engenharia de software com o cuidado com a segurança desde o início do ciclo. Os resultados das inserções destas práticas nos ciclos de vida de desenvolvimento foram positivos, como o estudo e o entendimento de ameaças, falhas e vulnerabilidades comuns, a definição de segurança como objetivo, e o desenvolvimento de testes de risco minuciosos e objetivos (MCGRAW, 2004).

Também com o intuito de aplicar as melhores práticas de segurança, organizações lançaram processos com o objetivo de incluir a segurança na etapa de design de aplicações, alguns exemplos desses processos são: o *Security Development Lifecycle* (SDL)

proposto pela empresa Microsoft, o *Software Assurance Maturity Model* (SAMM) proposto pela OWASP como um modelo de maturidade, o *Building Security in Maturity Model* (BSIMM) (GEER, 2010).

2.3.1 Ciclo de Vida de Desenvolvimento Seguro

O ciclo de vida de desenvolvimento seguro (*Security Development Lifecycle* - SDL) proposto pela Microsoft, compartilhado pela primeira vez em 2008, consiste em conjunto de práticas que auxiliam na garantia de segurança e cumprimento de requisitos. Os objetivos do SDL são reduzir os custos do desenvolvimento, pela redução do número e gravidade de vulnerabilidades (Microsoft, 2012).

De acordo com a documentação do SDL disponibilizada pela empresa Microsoft, as doze práticas propostas são (Microsoft, 2012):

1. **Providenciar Treinamento** - Todos os membros da equipe devem ter um conhecimento mínimo sobre segurança e sobre como atribuir segurança ao software e serviços para construir produtos mais seguros sem que haja prejuízo às necessidades de negócio e à entrega de valor ao usuário.

2. **Definir Requisitos de Segurança** - Definir os requisitos de segurança na fase inicial de design e planejamento do software é crucial para integrar a segurança de forma eficiente e minimizar interrupções. Entretanto, é importante atualizá-los continuamente para refletir mudanças na funcionalidade necessária e no cenário de ameaças.

3. **Definir Métricas e Relatório de Conformidade** - É necessária a definição dos níveis mínimos aceitáveis de qualidade de segurança e a garantia de que os critérios definidos serão cumpridos.

4. **Performar Modelagem de Ameaças** - Deve ser performada em ambientes nos quais existem riscos de segurança significantes. Pode ser aplicada em diferentes níveis, desde o nível de componente ao nível de sistema.

5. **Estabelecer Requisitos de Design** - Os requisitos de *design* devem ser definidos de forma consistente e com conhecimento sobre a proteção que fornecem. Pois, em muitos casos, a complexidade da implementação de funcionalidades de segurança é tão elevada que algumas escolhas de *design* tendem a gerar vulnerabilidades.

6. **Definir e Usar Padrões de Criptografia** - É ideal desenvolver padrões claros de criptografia que forneçam detalhes sobre cada elemento da implementação. Porém, devido ao nível de complexidade, deve ser feito por especialistas.

7. **Gerenciar o Risco de Segurança de Utilizar Componentes de Terceiros** - É fundamental a compreensão do impacto de uma vulnerabilidade de segurança nos componentes integrados no sistema. Com o objetivo de mitigar esse risco, faz-se necessário ter uma lista dos componentes de terceiros e um plano de resposta para novas vulnerabilidades descobertas. No entanto, é importante considerar validações adicionais, dependendo do tipo de componente utilizado e do potencial impacto.

8. **Usar Ferramentas Aprovadas** - Definir e publicar uma lista de ferramentas aprovadas com suas verificações de segurança.

9. **Performar Testes de Segurança de Análise Estática (SAST)** - Um método altamente escalável de revisão de segurança de código é a análise estática, a qual ajuda na garantia de que as regras de segurança de desenvolvimento estão sendo seguidas.

10. **Performar Testes de Segurança de Análise Dinâmica (DAST)** - Método que tem como objetivo monitorar o comportamento da aplicação em determinados cenários, como: problemas de privilégio de usuários e ataques "*fuzzing*".

11. **Performar Testes de Penetração** - Teste de penetração é um teste que deve ser performado por especialistas da área de segurança e que simula as ações de um *hacker*. O objetivo é a identificação de vulnerabilidades que não foram identificadas por outros métodos.

12. **Estabelecer um Padrão de Processo de Resposta a Incidente** - A elaboração de um padrão de resposta a incidente é de extrema importância para facilitar o endereçamento de novas ameaças que podem surgir. O padrão deve incluir quem deve ser acionado em situação de emergência de segurança e o protocolo para o serviço de segurança. Também é importante, testar o plano antes de ser necessário.

2.3.2 SAMM

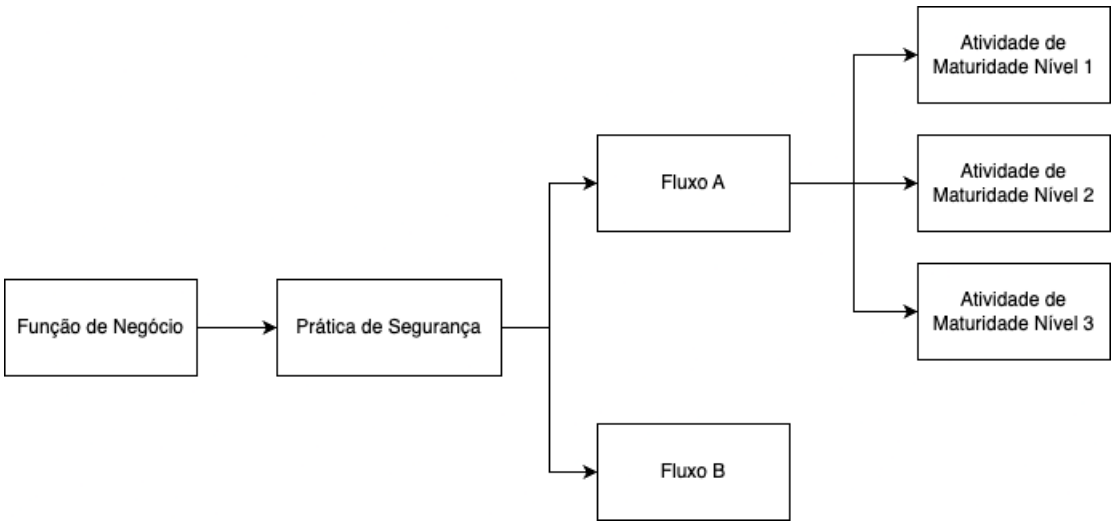
O SAMM proposto pela OWASP é um modelo de maturidade flexível e normativo que tem como objetivo a construção de segurança na organização do desenvolvimento de software de forma auto-avaliativa (RANSOME; MISRA, 2013). E por meio dessa auto-avaliação é possível saber em que estágio do trajeto rumo à segurança de software a organização está e o que é recomendado para a conquista do próximo nível de maturidade.

O modelo tem três características principais que são: os níveis de maturidade definidos em práticas de segurança, os trajetos claros para melhoria dos níveis de maturidade e também a não limitação a uma tecnologia, processo ou organização. Uma outra característica é que cada empresa ou organização pode definir os níveis de maturidade de

acordo com os seus objetivos, pois o SAMM não insiste que seja alcançado o maior nível de maturidade em todas categorias.

A estrutura base do modelo é apresentada na Figura 1, partindo da função de negócio, então dividindo em práticas de segurança, as quais possuem fluxos que resultam em três níveis de maturidade (The OWASP Foundation, 2020).

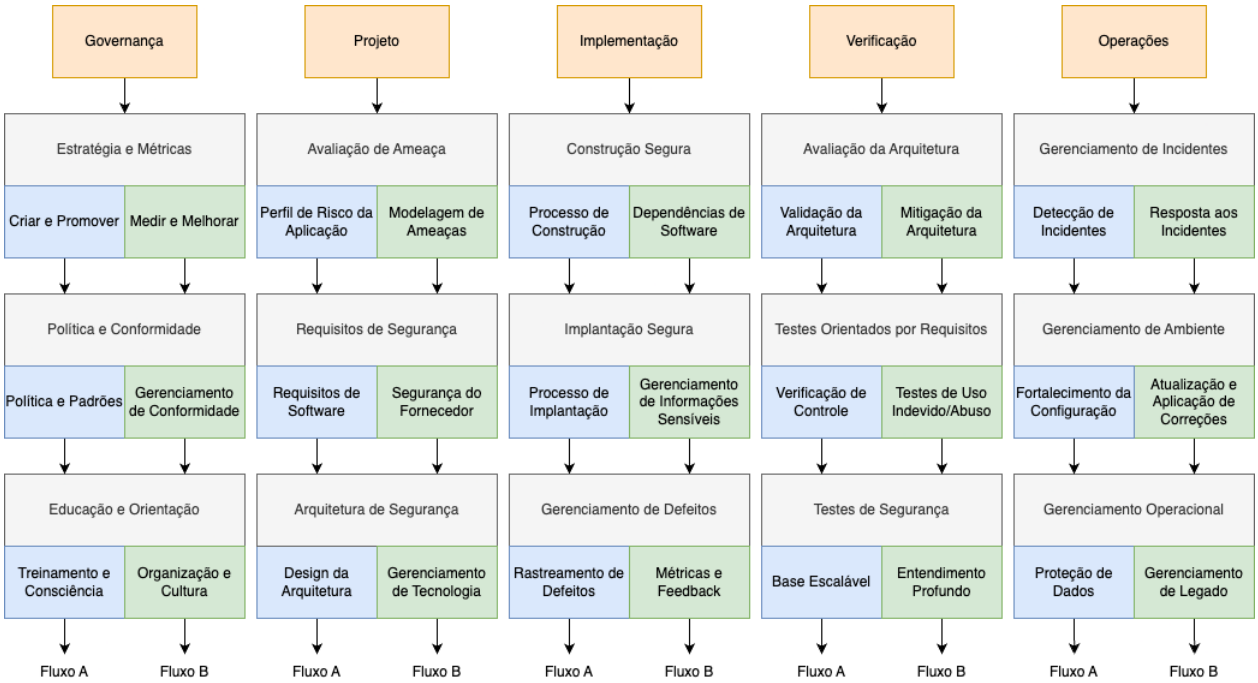
Figura 1 – Estrutura base de uma função de negócio com suas práticas de segurança e atividades.



Fonte: Adaptada (The OWASP Foundation, 2020)

Na Figura 2, apresenta-se o modelo de maturidade SAMM dividido em cinco funções de negócio (Governança, Projeto, Implantação, Verificação e Operações). Cada função possui três práticas de segurança. Cada prática possui atividades que são agrupadas em dois fluxos lógicos, estruturadas em três níveis de maturidade.

Figura 2 – Modelo SAMM com funções de negócio, práticas de segurança, fluxos e atividades.



Fonte: Adaptada (The OWASP Foundation, 2020)

2.4 Software de Código Aberto

A definição de programa de código aberto (*open source software* - OSS) pode ser compreendida a partir do o documento "*The Open Source Definition*" (A Definição do Código Aberto) da Iniciativa de Código Aberto (*Open Source Iniciative* - OSI) (Open Source Iniciative, 2006). O documento apresenta os seguintes dez requisitos necessários para um software ser considerado de código aberto:

1. **Redistribuição Livre** - Deve ser possível a venda ou fornecimento gratuito do software como parte de uma distribuição de software agregada que contém programas de diferentes fontes. Além disso, não deve ser exigido pagamento de *royalties* ou outras taxas pela venda.

2. **Código fonte** - É necessário um meio de acesso ao código-fonte, seja pela distribuição em conjunto com o produto ou até mesmo por *download* via *internet* sem custo. Além disso, o código-fonte deve ser claro e deve ser a maneira pela qual é feita alguma modificação no programa. Também, deve ser permitida a distribuição tanto em forma de código-fonte quanto forma compilada.

3. **Trabalhos derivados** - Modificações e trabalhos derivados devem ser permitidos pela licença, assim como a distribuição deles sob termos e condições da licença do

software original.

4. **Integridade do código fonte do autor** - É obrigatória a permissão de distribuição de programa derivado de código-fonte modificado e também a distribuição de "arquivos de correção" junto ao código-fonte. Porém, há também a possibilidade de restrição de distribuição do código-fonte modificado e também a exigência de nome ou número de versão do trabalho derivado diferentes do programa original.

5. **Não discriminação contra pessoas ou grupos** - Não é permitida a discriminação por parte da licença contra pessoas ou grupos.

6. **Não discriminação contra áreas de empreendimento** - A licença não deve restringir o uso do programa a uma área específica.

7. **Distribuição de Licença** - Os direitos relacionados ao programa devem ser estendidos a quem o programa for redistribuído, sem exigir a obtenção de uma licença adicional.

8. **Licença não deve ser específica para um produto** - A licença não deve ser condicionada ao programa fazer parte de uma distribuição de software específica. Em caso de utilização ou redistribuição de acordo com os termos da licença, aqueles que recebem a redistribuição devem ter os mesmos direitos concedidos na distribuição original do programa.

9. **Licença não deve restringir outro software** - Não devem ser aplicadas restrições aos softwares que são distribuídos com o software da licença.

10. **Licença deve ser neutra em relação a tecnologia** - A licença não deve ser baseada em uma tecnologia ou estilo de interface.

Um software que cumpre com todos esses requisitos pode ser considerado um software de código aberto, porém é importante ressaltar que há diferença entre software de código aberto e software livre. O software livre deve cumprir quatro requisitos, que pela comunidade são chamados de quatro liberdades, as quais são enumeradas a partir do zero (Free Software Foundation, 2019):

1. **Liberdade 0** - Liberdade para executar o programa de acordo com qualquer propósito.

2. **Liberdade 1** - Por ter acesso ao código-fonte, o qual é um pré-requisito para essa liberdade, deve haver liberdade para estudar o funcionamento do software e alterar de acordo com o que deseja.

3. **Liberdade 2** - Liberdade para redistribuição de cópias com o intuito de ajudar os outros.

4. **Liberdade 3** - Liberdade de distribuição de cópias modificadas com o pré-requisito de acesso ao código fonte.

Um software pode ser de código aberto e livre, porém também é possível que um software seja apenas de código aberto ou apenas um software livre. Importante ressaltar que software livre, não necessariamente é um software gratuito (Free Software Foundation, 2019). Com o objetivo de facilitar a identificação de licenças de software de código aberto, a OSI disponibiliza uma lista das licenças de código aberto (Open Source Iniciative, 2022), a qual pode ser acessada no *site* da iniciativa.

A disponibilização do código-fonte é essencial para o software de código aberto, como foi apresentado no segundo requisito da listagem da OSI (Open Source Iniciative, 2006). Algumas plataformas como *SourceForge* [1], *BitBucket*[2] da empresa Atlassian e o *GitHub*[3] tem como funcionalidades a disponibilização de código fonte e a contribuição em projetos.

A principal plataforma utilizada pelas comunidades de OSS é o GitHub, que foi lançado em 2008 (WANSTRATH, 2008), onde é possível acompanhar as atividades de outros desenvolvedores e da comunidade em geral, além da possibilidade de se inscrever para receber as notificações daquele projeto (THUNG et al., 2013).

No GitHub, os projetos são chamados de repositórios, os quais possuem um endereço *Uniform Resource Locator* (URL) para o acesso, com a possibilidade de serem privados ou públicos. Nos repositórios é possível ter acesso ao código, além de funcionalidades como revisão de código, gerenciamento de tarefas, entre outras.

Existem também algumas maneiras de verificar a atividade de uma comunidade em um projeto de código aberto como pelo número de tarefas criadas recentemente, pelas interações nas revisões de código ou pelo número de "*forks*" (PETERSON, 2013), que são cópias dos repositórios para fazer alterações que podem ser adicionadas ao projeto original.

2.4.1 Estrutura das Comunidades de Código Aberto

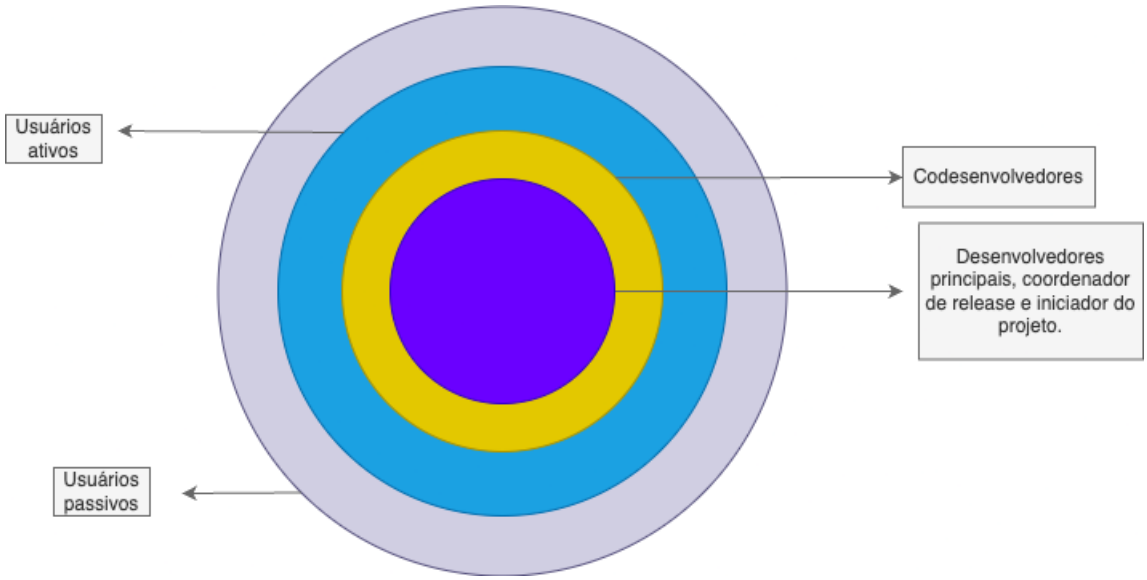
A Figura 3 apresenta uma estrutura de uma equipe de desenvolvimento de um software livre de código aberto (*Free/Libre Open Source Software* - FLOSS), porém a estrutura também pode ser considerada para OSS. Ela é dividida basicamente em quatro

[1] https://sourceforge.net/
[2] https://bitbucket.org/
[3] https://github.com/

níveis de participação e interação, quanto mais interno o círculo maior a contribuição. Os desenvolvedores principais, coordenador de versões e o iniciador ou conjunto de iniciadores dos projetos são aqueles que tem maior conhecimento do código e do projeto (CROWSTON; HOWISON, 2005). Importante ressaltar que nem sempre uma comunidade de OSS possui exatamente essas funções definidas, podendo não ter um coordenador de *release*, por exemplo.

Os codesenvolvedores são contribuidores que já possuem certo conhecimento sobre o código e projeto, logo conseguem contribuir com submissão de correções de defeitos ou modificação ou revisão de código (ALMARZOUQ et al., 2005). Há também os diferentes tipos de usuários: passivos e ativos. Os usuários ativos são aqueles que interagem com a comunidade, sem o desenvolvimento de código, ou seja, por meio de avisos de defeitos, ideias de funcionalidades etc. Já os usuários passivos estão apenas interessados em usar o produto final sem interagir com a comunidade (CROWSTON; HOWISON, 2005).

Figura 3 – Estrutura de equipe de desenvolvimento de OSS livre.



Fonte: Adaptada (CROWSTON; HOWISON, 2005).

2.5 Segurança em Software de Código Aberto

A Fundação de Segurança de Código Aberto (*Open Source Security Foundation* - OpenSSF) é uma organização da Linux Foundation (Open Source Security Foundation, 2021b) que tem como principal objetivo a promoção da segurança de código aberto para todos de forma colaborativa com as comunidades envolvidas, desde o desenvolvimento até a implementação (Open Source Security Foundation, 2023). Para isso, a OpenSSF agrega iniciativas importantes de segurança de código aberto de diversas fontes.

A OpenSSF visa um ecossistema de código aberto com compartilhamento e uti-

lização de software de alta qualidade. Além disso, também visa o tratamento natural e proativo da segurança, o que inclui a aprendizagem simplificada de práticas de desenvolvimento seguro com o auxílio de ferramentas para a aplicação das práticas de forma automatizada (Open Source Security Foundation, 2023).

Também faz parte desse ecossistema a criação e distribuição de políticas de segurança por desenvolvedores auditores e reguladores, juntamente com eficiente identificação e solução de problemas de segurança. Para que seja possível alcançar esse ecossistema objetivo, a organização provê ferramentas, treinamentos, serviços e recursos (Open Source Security Foundation, 2023). Um desses recursos é o Programa de Selo de Melhores Práticas da OpenSSF.

2.5.1 Programa de Selo de Melhores Práticas da OpenSSF

A plataforma GitHub utiliza distintivos com a função de apresentar o status de um fluxo de trabalho (GitHub, 2023). Exemplos de distintivos são distintivos de licença, cobertura de código por testes e qualidade de código. Normalmente, ele é adicionado no arquivo README.md do repositório, o qual é o arquivo de apresentação das informações principais sobre o projeto.

O Programa de Selo de Melhores Práticas da OpenSSF foi desenvolvido com o objetivo de fomentar a aplicação das melhores práticas pelos projetos de FLOSS, além do auxílio na identificação de projetos que fazem essa aplicação. O programa une as melhores práticas para FLOSS em um sistema de distintivo. O sistema é chamado “BadgeApp”, no qual é possível a auto-certificação do atendimento de um projeto aos critérios e a exibição do resultado em forma de distintivo (Open Source Security Foundation, 2023).

O PSMPO apresenta 67 práticas divididas em 6 conjuntos (Open Source Security Foundation, 2023):

1. **Básicas** - Apresenta 13 práticas divididas em 4 subconjuntos. As práticas pertencentes a esse conjunto estão relacionadas a informações sobre projeto, documentação e licença FLOSS.

2. **Controle de alteração** - Apresenta 9 práticas divididas em 3 subconjuntos. As práticas pertencentes a esse conjunto estão relacionadas ao controle de versão do código fonte e às notas de versão.

3. **Reportagem** - Apresenta 8 práticas divididas em 2 subconjuntos. As práticas pertencentes a esse conjunto estão relacionadas ao relatório de defeitos e vulnerabilidades.

4. **Qualidade** - Apresenta 13 práticas divididas em 4 subconjuntos. As práticas pertencentes a esse conjunto estão relacionadas à aplicação de testes automatizados, testes

de novas funcionalidades, bandeiras de aviso sobre a compilação e à necessidade de um sistema de compilação que funcione.

5. **Segurança** - Apresenta 16 práticas divididas em 5 subconjuntos. As práticas pertencentes a esse conjunto estão relacionadas ao conhecimento do desenvolvimento seguro, ao uso de boas práticas de criptografia, à proteção a ataques do modelo "*homem no meio*" e à solução de vulnerabilidades conhecidas.

6. **Análise** - Apresenta 8 práticas divididas em 2 subconjuntos. As práticas pertencentes a esse conjunto estão relacionadas à aplicação de análise estática e análise dinâmica do projeto.

O PSMPO incorpora diversas práticas comuns a modelos e ciclos de desenvolvimento seguro (Open Source Security Foundation, 2023). Um exemplo notável é a inclusão de um subconjunto de práticas voltadas para o uso fundamental de boas práticas de criptografia no âmbito das práticas de segurança. Este subconjunto pode ser equiparado à prática "*definir e usar padrões de criptografia*" do SDL (Microsoft, 2012). Similarmente, o SAMM apresenta uma correspondência na área de Gerenciamento de Defeitos, especificamente na atividade de Métricas e Feedback (The OWASP Foundation, 2020), essa atividade pode ser analogamente comparada ao subconjunto do PSMPO que aborda as análises, estática e dinâmica, nesse contexto específico.

Para a auto-certificação é necessário preencher um formulário sobre as práticas do projeto em áreas como controle de mudanças, qualidade, segurança e análise. O projeto pode receber como classificação um percentual que representa o quanto dos critérios para alcançar o nível "passando"(*passing*) foi atingido. Caso já tenha alcançado o nível "passando", pode ser classificado também em prata ou ouro, a depender do atendimento dos critérios para alcançar esses níveis (Open Source Security Foundation, 2021a).

Na plataforma do PSMPO estão registrados mais de cinco mil e novecentos projetos, dos quais mais de mil já alcançaram o nível "passando". Enquanto, apenas cerca de dois mil e seiscentos projetos aplicam no mínimo 25% das práticas do programa (Open Source Security Foundation, 2021a).

É importante ressaltar que os níveis apresentados pelo distintivo servem apenas como um indicador de aplicação das melhores práticas, portanto nem mesmo o nível "ouro" garante a inexistência de vulnerabilidades no código ou a perfeição dos processos de desenvolvimento (The Linux Foundation, 2020).

2.6 Análise Exploratória de Dados

A análise exploratória de dados é um estágio da análise de dados, a qual não necessariamente precisa trabalhar com probabilidade, significância ou confiança, porém quando

possui muitos dados, não é preciso lidar com todos, mas uma amostra (TUKEY, 1972). Também a análise exploratória de dados abrange estudo da correlação entre preditores, e entre preditores (ou variável preditora, é uma variável que influencia a variável-alvo) e uma variável-alvo em muitos projetos de modelagem (BRUCE; BRUCE, 2019).

2.6.1 Correlação

Correlação é uma forma de medição da direção e intensidade da relação entre duas variáveis quantitativas, sendo geralmente representada pela letra r (MOORE, 2009). A correlação pode ser positiva quando valores altos de ambas variáveis se acompanham ou negativa quando os valores baixos de uma variável acompanha os valores altos de outra (BRUCE; BRUCE, 2019).

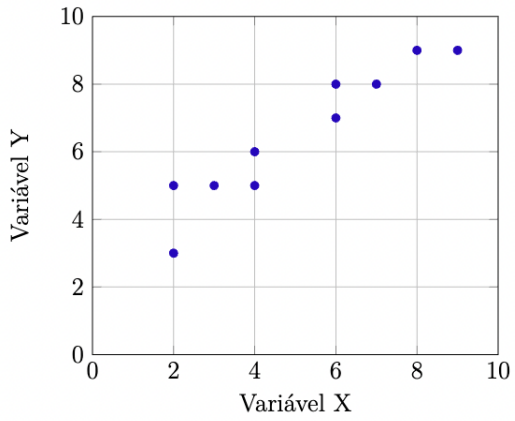
Para estimar a correlação entre duas variáveis na mesma escala é possível utilizar o *Coeficiente de Correlação*, o qual é uma métrica que indica o quão ligadas são duas ou mais variáveis numéricas, a qual vai de -1 a +1, com o 0 sendo um indicador de ausência de correlação (BRUCE; BRUCE, 2019).

Em situações nas quais não há informações sobre a distribuição dos dados, pode ser utilizado o *Coeficiente de Correlação de Spearman* por ser um teste não paramétrico. Para calcular o coeficiente utilizamos a Equação 2.1, na qual ρ representa o coeficiente de correlação de Spearman, d_i representa a diferença entre as posições dos pares de classificações e n é o número total de pares de classificações.

$$\rho = 1 - \frac{6 \sum d_i^2}{n(n^2 - 1)} \tag{2.1}$$

Para visualização e investigação (TUKEY, 1972) da relação entre duas variáveis de dados mensuradas, geralmente é utilizado um gráfico de dispersão, no qual o eixo X representa uma variável, e o eixo Y representa outra, logo cada ponto no gráfico representa um registro (BRUCE; BRUCE, 2019) como é possível visualizar na Figura 4.

Figura 4 – Exemplo de gráfico de dispersão.



Fonte: Autor.

3 Metodologia

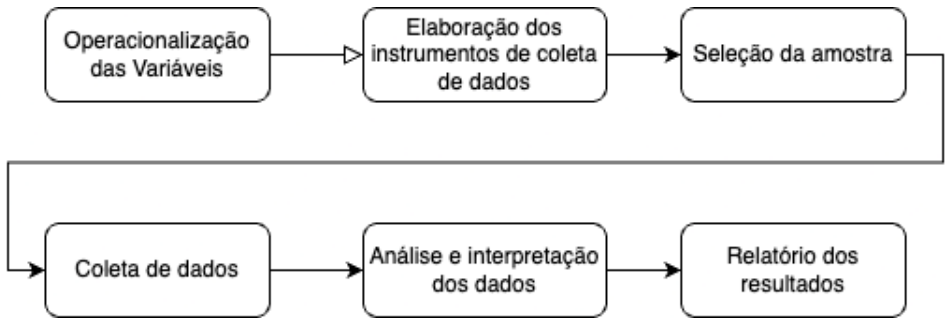
3.1 Considerações Iniciais do Capítulo

Neste capítulo apresenta-se o plano metodológico adotado neste trabalho. O Planejamento da Pesquisa está dividido em seis etapas: operacionalização das variáveis, elaboração dos instrumentos de coleta de dados, seleção da amostra, coleta de dados, análise e interpretação dos dados e relatório dos resultados.

3.2 Planejamento da Pesquisa

A Figura 4 apresenta um diagrama com as seis etapas deste trabalho, com o intuito de facilitar o acompanhamento do fluxo da pesquisa (GIL, 2002).

Figura 5 – Diagrama da pesquisa.



Fonte: Autor.

3.2.1 Operacionalização das Variáveis

A *Densidade de Vulnerabilidades (d)* é calculada ao dividir o número de vulnerabilidades (v) encontradas no projeto pelo número de linhas de código dividido por mil ($kloc$), como apresentado na Equação 3.1.

$$d = (\frac{v}{kloc}) \tag{3.1}$$

O cálculo da *Densidade de Code Smells (c)* é feito de maneira análoga ao cálculo da Densidade de Vulnerabilidades, porém com a substituição do número de vulnerabilidades pelo número de *code smells (nc)* encontrados no projeto, como apresentado na Equação 3.2.

$$c = (\frac{nc}{kloc}) \tag{3.2}$$

A Equação 3.3 apresenta como é calculada a *Densidade de Hotspots (h)*, onde número de *hotspots* encontrados no projeto é dividido pelo número de linhas de código dividido por mil (*kloc*).

$$h = (\frac{nh}{kloc}) \tag{3.3}$$

O *Número de Práticas* do PSMPO aplicadas por cada OSS é fornecido pela base de projetos do site *BadgeApp* da OpenSSF (Open Source Security Foundation, 2021a). A variável será representada pela sigla *np* (número de práticas) e pode variar entre 0 e 67, pois 67 é o número máximo de práticas apresentadas pelo PSMPO.

3.2.2 Elaboração dos Instrumentos de Coleta de Dados

A pesquisa deste trabalho requer a coleta de dados de forma organizada da base de projetos do PSMPO. Como não é possível a exportação dos dados pelo site, tornou-se necessário o desenvolvimento de um conjunto de *scripts* de *web scraping* para uma coleta de uma amostra maior e de maneira mais rápida.

A técnica de *web scraping* consiste em um processo sistemático de extração e junção de informações de interesse retirados da *Web* (GLEZ-PEñA et al., 2014). Os *scripts* irão aplicar essa técnica no site do PSMPO para coletar os repositórios dos projetos e quantas e quais práticas do PSMPO aplicam.

Também são necessários *scripts* para coleta das métricas *kloc* (número de linhas de código), número de vulnerabilidades, número de *code smells* e número de *hotspots* de cada projeto. Para isso, os scripts devem executar uma análise estática utilizando a plataforma *SonarQube* localmente e a ferramenta *SonarScanner*.

3.2.3 Seleção da Amostra

A amostra de projetos é composta pelos OSSs encontrados na base de projetos do PSMPO e que armazenam o código fonte no *GitHub*, uma vez que o script de análise estática somente consegue acessar repositórios na plataforma indicada.

Outros requisitos dos projetos que constituem a amostra selecionada são o percentual mínimo de 80% de bytes de código em linguagem suportada pelo SonarQube na edição Community e o mínimo de mil linhas de código identificadas pelo SonarQube.

As linguagens suportadas pela edição Community do SonarQube na versão 10.2 (SonarSource S.A, 2023a) são Azure Resource Manager, CloudFormation, C#, CSS, Docker, Flex, Go, HTML, Java, JavaScript, Kotlin, Kubernetes, PHP, Python, Ruby, Scala, Secrets, Terraform, TypeScript, VB.NET e XML.

3.2.4 Coleta de Dados

Esta etapa tem como objetivo reunir informações sobre os projetos de código aberto encontrados na base de projetos do Programa de Selo de Melhores Práticas da OpenSSF (PSMPO) que possuem análise estática. As informações buscadas são a densidade de vulnerabilidades de cada projeto, a densidade de *code smells*, a densidade de *hotspots* de segurança e quais as práticas do PSMPO são aplicadas.

Primeiramente, é feita a coleta da quantidade de práticas do PSMPO aplicadas em cada projeto da amostra e quais são as práticas. Esta coleta é feita pelos *scripts* de *web scraping*.

Na segunda fase da coleta, é verificada a análise estática com a ferramenta *SonarQube* em cada projeto. As informações que devem ser obtidas são a quantidade de linhas de código, quantidade de vulnerabilidades, quantidade de *code smells* e quantidade de *hotspots*, para que sejam calculadas as densidades de vulnerabilidades, de *code smells* e de *hotspots*.

Os dados necessários da análise estática são obtidos e organizados também pelos *scripts* citados na seção *Elaboração dos Instrumentos de Coleta de Dados*.

3.2.5 Análise de Dados

Nesta etapa, os dados são organizados em tabelas e gráficos de dispersão. São calculados os coeficientes de correlação entre as variáveis d (Densidade de Vulnerabilidade) e np (Número de Práticas), entre as variáveis c (Densidade de *Code Smells*) e np, e entre as variáveis h (Densidade de *Hotspots*) e np, de acordo com a equação 2.1 apresentada no Capítulo 2 - *Segurança de Software*.

É analisada a correlação das densidades de vulnerabilidades, *code smells* e *hotspots* com o total de práticas do PSMPO, como também com segmentos de práticas relacionadas, de forma a buscar entender o impacto por tipo de prática.

Com o resultado do cálculo, será possível verificar se há uma correlação entre as variáveis e com o auxílio dos gráficos será possível apresentar visualmente a relação entre as variáveis.

3.2.6 Relatório

Os resultados obtidos através da análise dos dados sobre as correlações são detalhadamente expostos na seção de Conclusão, onde também se destacam os resultados esperados, possibilitando uma comparação entre ambos.

4 Coleta de Dados

A etapa de Coleta de Dados seguiu o que foi proposto na Metodologia e pode ser dividida em duas etapas, a Coleta de Práticas e a Coleta de Densidade.

4.1 Coleta de Práticas

A coleta das respostas do formulário do PSMPO sobre as práticas de cada projeto foi feita de forma automatizada com um script de *web scraping*, o qual agrupou as seguintes informações por projeto: identificador do projeto na base do Badge, o endereço do repositório no GitHub e as 67 respostas para perguntas sobre as práticas.

4.1.1 Coleta de identificadores do projetos

Nesta subetapa, foram coletados os identificadores de cada projeto presente na base do PSMPO e organizados em um arquivo no formato CSV. Os identificadores dos projetos são necessários para acessar o endereço URL do projeto, onde estão as informações de forma detalhada. Até o momento em que foi executado o script, haviam 198 páginas, o que resultou em 5922 projetos identificados.

4.1.2 Coleta das práticas

Nesta subetapa, os 5922 identificadores dos projetos, foram utilizados para montar o endereço URL de cada um, logo foram coletadas informações como endereço de armazenamento do código e as respostas de cada prática, pois com a aplicação da técnica de web scraping foi possível analisar cada elemento HTML de cada página de maneira automatizada e armazenar as informações em um arquivo CSV.

As informações básicas sobre as práticas coletadas que podem ser extraídas são apresentadas na Tabela 1, em que as práticas estão divididas pelas categorias e também são apresentados os dados gerais. A média de práticas por projeto é baixa, pois pode ser arredondada para 26 práticas, porém o PSMPO apresenta 67, portanto representa menos de 40% do total.

A pior média é apresentada pela categoria de análise, a qual possui apenas 8 práticas relacionadas a análise estática e dinâmica do projeto. A maior média é apresentada pelas práticas básicas, pois são práticas essenciais para projetos de código aberto.

É importante observar também que a média de práticas de segurança é baixa, com uma média de 3,561 de um total de 16 práticas previstas. Além disso, a mediana está em

zero, indicando que grande parte dos projetos da amostra não realizada nenhuma prática de segurança.

Tabela 1 – Tabela de estatística descritiva das métricas.

Categoria	Valor Máximo	Valor Mínimo	Média	Mediana	Desvio Padrão
Básicas	13	0	8,692	8	3,595
Controle de Mudança	9	0	4,904	3	2,752
Relatório	8	0	2,517	0	3,240
Qualidade	13	0	4,360	2	5,219
Segurança	16	0	3,561	0	5,332
Análise	8	0	1,673	0	2,654
Geral	67	0	25,709	13	20,852

Fonte: Autor.

4.1.3 Verificação de armazenamento de código

Nesta subetapa, foi feita a verificação da plataforma em que o código está armazenado, caso a plataforma seja diferente do GitHub, é descartada, pois a *Application Programming Interface* (API) do GitHub possibilita a verificação das linguagens utilizadas em cada projeto. O descarte resultou em uma diminuição do conjunto de projetos, de 5922 para 5333, dentre os projetos descartados, houve o descarte manual de 9 projetos que estavam armazenados no GitHub, porém não estavam com as práticas preenchidas na plataforma do PSMPO.

4.2 Coleta de Métricas de Código

Nesta etapa o principal objetivo foi coletar o número de linhas de código de cada projeto (*nloc*), número de vulnerabilidades (*v*), número de *code smells* (*nc*) e número de *hotspots* (*nh*) identificados pelo SonarQube. Para isso, foram necessárias três subetapas.

4.2.1 Coleta de linguagens de cada projeto

Nesta subetapa, foram feitas requisições para a API do GitHub para coleta das linguagens utilizadas em cada projeto, com o objetivo de possibilitar o próximo passo desta etapa.

Foi feita a verificação se ao menos 80% dos bytes de código de cada projeto são em linguagens suportadas pela edição Community do SonarQube. Esta verificação resultou na redução do número de projetos de 5333 para 3025.

Há uma predominância de projetos nas linguagens Go, Java e Python, como é observável na Tabela 2. Na tabela estão listadas as 20 linguagens com maior percentual dos bytes dos projetos da amostra.

Tabela 2 – Tabela de percentual de bytes por linguagem das 20 linguagens mais presentes.

Linguagem	Percentual dos bytes do total identificado
Go	26.18%
Java	25.18%
Python	10.33%
PHP	9.18%
JavaScript	8.98%
TypeScript	6.13%
HTML	4.74%
C#	2.50%
Ruby	1.23%
CSS	1.14%
Scala	0.65%
Shell	0.64%
Kotlin	0.60%
SCSS	0.29%
C	0.22%
Vue	0.20%
Makefile	0.13%
Twig	0.10%
PLpgSQL	0.10%
C++	0.10%

Fonte: Autor.

4.2.2 Coleta de nloc, números de vulnerabilidades, code smells e hotspots

Nesta subetapa, foi coletado o número de linhas de código, o número de vulnerabilidades, o número de *code smells* e o número de *hotspots* de cada projeto com o auxílio do SonarScanner, o qual é um escaneador de projetos, juntamente com o SonarQube. Durante a coleta, foram descartados 55 projetos, por problemas durante o escaneamento, causados pela configuração generalista necessária para a vasta extensão de linguagens encontradas nos projetos, o que resultou ao final desta subetapa o número de 2970 projetos.

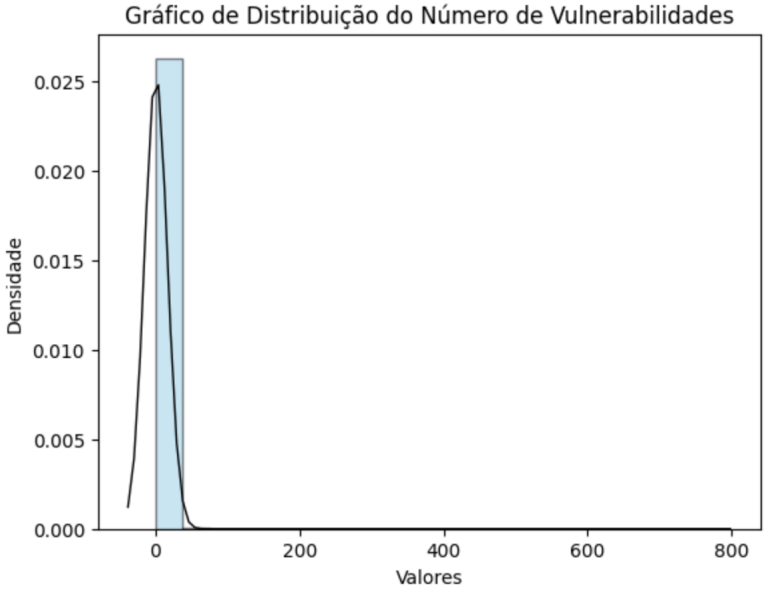
Tabela 3 – Tabela de estatística descritiva das métricas.

Métrica	Valor Máximo	Valor Mínimo	Média	Mediana	Desvio Padrão
Hotspots	8731	0	45,483	2	285,938
Code Smells	158633	0	979,987	51	6422,778
Vulnerabilidades	760	0	0,807	0	15,738

Fonte: Autor.

A distribuição do número de vulnerabilidades pode ser vista na Figura 6. Há um grande número de projetos com nenhuma vulnerabilidade como é notável pelo gráfico e pela Tabela 3, em que apresenta a média menor que 1 e a mediana como 0.

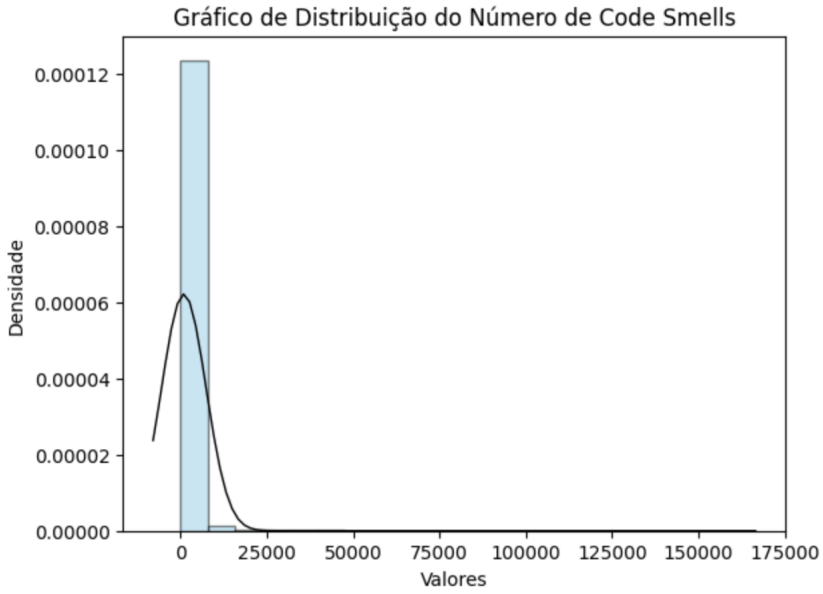
Figura 6 – Gráfico de distribuição do número de vulnerabilidades.



Fonte: Autor.

A distribuição do número de *code smells* se assemelha à distribuição do número de vulnerabilidades por ter uma concentração no início dos valores, como é visível na Figura 7. A média está em aproximadamente 980 *code smells*, porém com mediana de 51. O desvio padrão de cerca de 6422 indica uma grande variação dessa métrica entre os projetos.

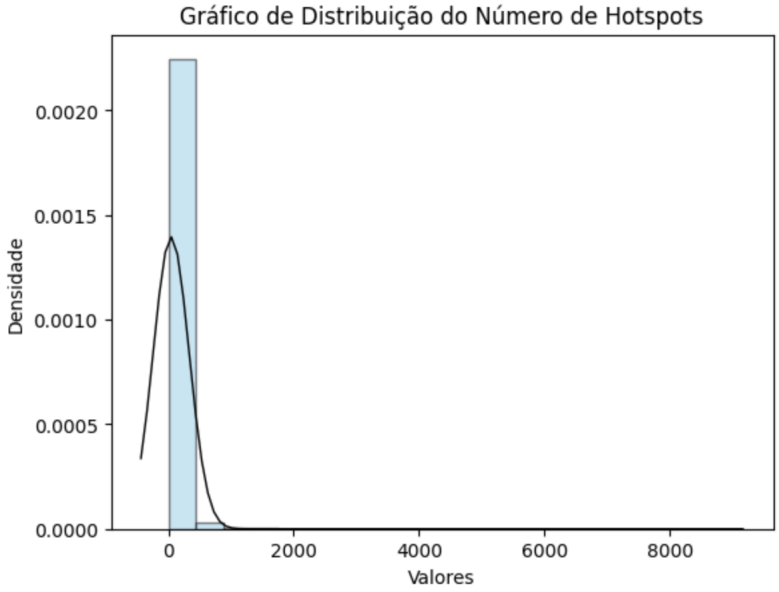
Figura 7 – Gráfico de distribuição do número de code smells.



Fonte: Autor.

O padrão assimétrico apresentado nas distribuições de *code smells* e vulnerabilidades ocorre na distribuição de *hotspots*, como observável na Figura 8. A mediana de valor 2 ajuda na compreensão da distribuição, pois os valores de *hotspots* em sua predominância são baixos, o que é notável pela média de 45,483 *hotspots* por projeto.

Figura 8 – Gráfico de distribuição do número de hotspots.



Fonte: Autor.

4.2.3 Verificação do número mínimo de linhas

Nesta subetapa, foram eliminados os projetos com menos de mil linhas de código, pois pelo baixo número de linhas, há uma maior possibilidade de ser um projeto pessoal ou voltado para documentação, o que poderia gerar ruído na análise dos dados. Portanto, dos 2970 projetos, após a redução foram mantidos 1791 projetos com os dados necessários para a análise.

A Tabela 4 apresenta que a média da métrica *kloc* é aproximadamente 45, o que representa 45 mil linhas de código. O menor projeto possui apenas 1006 linhas de código e o maior possui 2502085.

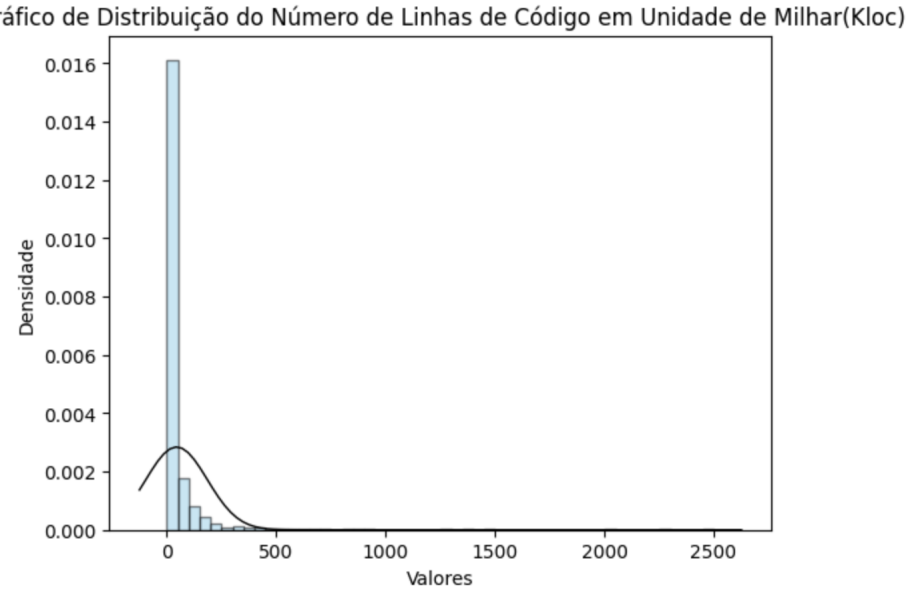
Tabela 4 – Tabela de estatística descritiva da métrica kloc.

Métrica	Valor Máximo	Valor Mínimo	Média	Mediana	Desvio Padrão
Kloc	2502,085	1,006	45,594	9,081	140,354

Fonte: Autor.

A Figura 9 é um gráfico de distribuição do número de linhas no modelo kloc, é notável uma grande concentração projetos com o número baixo da métrica. Essa concentração é perceptível também pela média e mediana apresentadas na Tabela 4, porém o gráfico se estende até próximo do valor 2500, por causa do valor máximo da amostra.

Figura 9 – Gráfico de distribuição do número de linhas de código em unidade de milhar.



Fonte: Autor.

5 Análise de Dados

A análise dos dados foi dividida em três etapas, cada etapa relacionada a uma métrica a ser analisada de maneira independente e também foi definida uma seção para análise das práticas do PSMPO.

5.1 Práticas

O PSMPO é estruturado em 6 categorias, cada uma com um conjunto de práticas. Cada categoria e o número de práticas estão apresentados na Tabela 5. A categoria de segurança é a categoria com o maior número de práticas, pois o PSMPO tem foco no aumento da segurança dos projetos.

Tabela 5 – Tabela de quantidade de práticas de cada categoria.

Categoria	Número de práticas
Básicas	13
Controle de alteração	9
Reportagem	8
Qualidade	13
Segurança	16
Análise	8

Fonte: Autor.

Cada prática do PSMPO é identificada por uma *tag*, as tags também foram utilizadas para identificar cada prática na Tabela 6, onde estão listadas as práticas de segurança na mesma ordem apresentada pelo PSMPO. A prática de segurança mais aplicada é a *no_leaked_credentials* que consiste no não vazamento de credenciais privadas que são usadas para limitar o acesso público em repositórios públicos (Open Source Security Foundation, 2021a).

Tabela 6 – Tabela de aplicação de práticas de segurança.

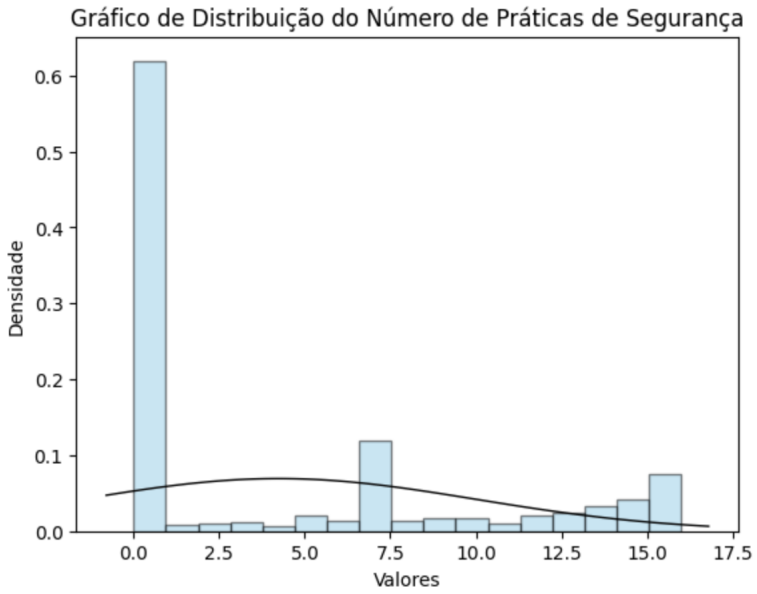
Prática	Número de projetos que aplicam a prática	Percentual
know_secure_design	640	35,97%
know_common_errors	644	36,20%
crypto_published	401	22,54%
crypto_call	401	22,54%
crypto_floss	407	22,87%
crypto_keylength	302	16,97%
crypto_working	346	19,44%
crypto_weaknesses	348	19,56%
crypto_pfs	226	12,70%
crypto_password_storage	206	11,57%
crypto_random	278	15,62%
delivery_mitm	659	37,04%
delivery_unsigned	634	35,63%
vulnerabilities_fixed_60_days	666	37,43%
vulnerabilities_critical_fixed	674	37,88%
no_leaked_credentials	700	39,34%

Fonte: Autor.

A prática de segurança menos aplicada é a *crypto_password_storage* que compreende se o software desenvolvido pelo projeto resultar no armazenamento de senhas para autenticação de usuários externos, é imprescindível que essas senhas sejam armazenadas como hashes iterativos, utilizando um "salt"único para cada usuário. Para esse propósito, recomenda-se a implementação de um algoritmo de "key stretching"(iteração), como o Argon2id, Bcrypt, Scrypt ou PBKDF2. Essa abordagem visa fortalecer a segurança do armazenamento das senhas e proteger a integridade das informações dos usuários (Open Source Security Foundation, 2021a).

A maioria dos projetos não aplica nenhuma prática de segurança, como é mostrado na Figura 10. O gráfico demonstra que mais de 60% dos projetos não aplicam nenhuma prática de segurança, mais de 10% aplica 7 práticas e apenas aproximadamente 7% dos projetos aplicam as 16 práticas de segurança.

Figura 10 – Gráfico de Distribuição de Práticas de Segurança.

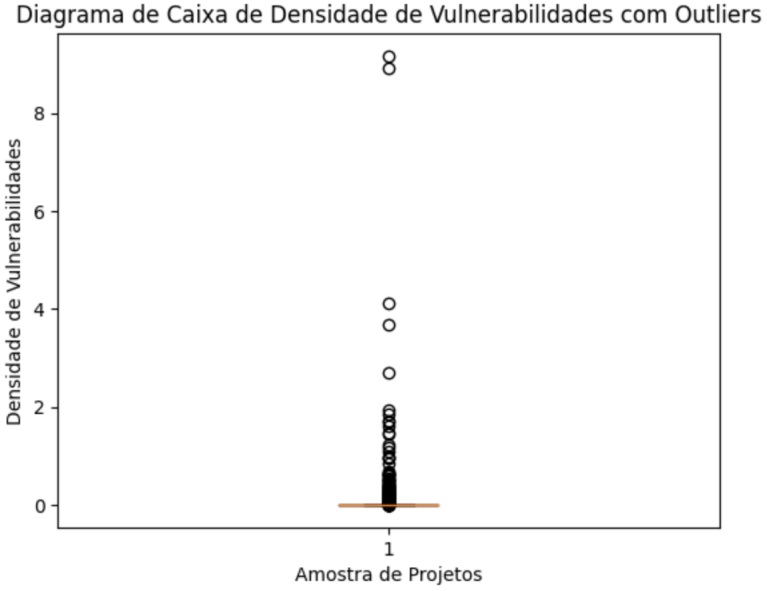


Fonte: Autor.

5.2 Densidade de Vulnerabilidades

A densidade de vulnerabilidades dos projetos da amostra foi apresentada em um diagrama de caixa, sem um tratamento prévio, para a compreensão inicial da variação dos valores. O diagrama pode ser visto na Figura 11.

Figura 11 – Diagrama de Caixa de densidade de vulnerabilidades com outliers.



Fonte: Autor.

Ao visualizar a Figura 11, é perceptível a presença de *outliers*, a mediana está indicada como 0, assim como o primeiro quartil *(Q1)* e o terceiro quartil *(Q3)*, de forma que os valores diferentes de 0, são representados como *outliers*. Para melhor compreensão da densidade de vulnerabilidades, foi aplicada a técnica de Tukey ou *boxplot* (HOAGLIN; IGLEWICZ, 1987), a qual utiliza a fórmula da Equação 5.1 para calcular o valor do interquartil *(IQR)*.

$$IQR = Q3 - Q1 \tag{5.1}$$

Após o cálculo do IQR é definido o limite inferior *(Linf)*, de acordo com a Equação 5.2, em que k é um parâmetro que pode ser ajustado.

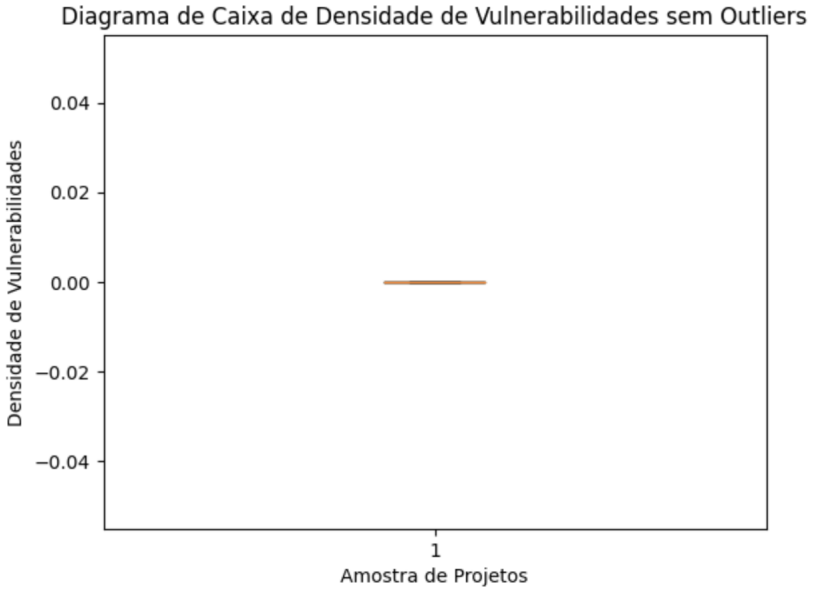
$$Linf = Q1 - k * IQR \tag{5.2}$$

O limite superior *(Lsup)* é definido de acordo com a Equação 5.3. Os dados anômalos ou *outliers* são os valores que ultrapassam os limites.

$$Lsup = Q3 + k * IQR \tag{5.3}$$

O valor para k utilizado foi 2,3, pois o valor padrão, o qual é 1,5 pode chegar a representar 25% da amostra de dados. Ao alterar o parâmetro k para 2,3 a probabilidade de que existem dados anômalos é de 5% (HOAGLIN; IGLEWICZ, 1987).

Figura 12 – Diagrama de Caixa de densidade de vulnerabilidades sem outliers.

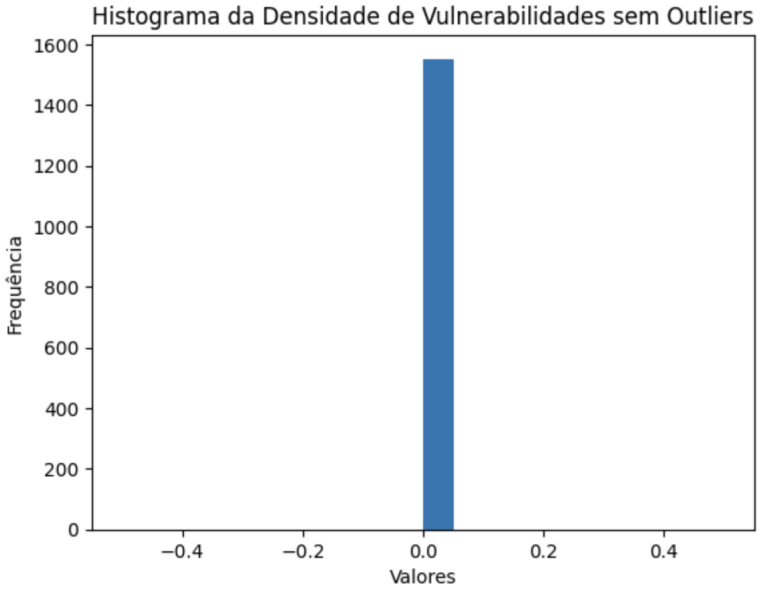


Fonte: Autor.

Como resultado da aplicação dessa técnica, houve a remoção dos *outliers*, porém como é perceptível na Figura 12, todos os valores diferentes de 0 foram considerados *outli-*

ers. Mais de 1500 projetos não foram considerados *outliers* e portanto, possuem densidade igual a zero, como é notável na Figura 13. Isto indica que mais de 80% da amostra não teve nenhuma vulnerabilidade identificada.

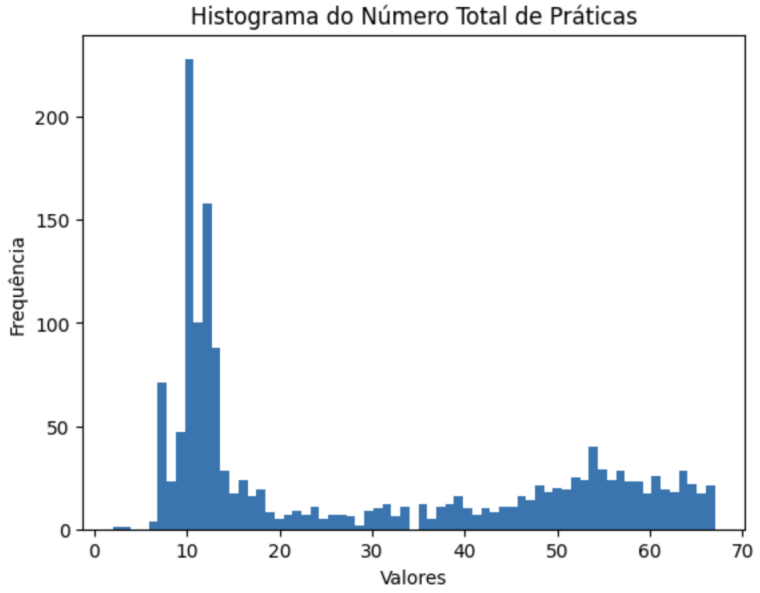
Figura 13 – Histograma da densidade de vulnerabilidades da amostra sem outliers.



Fonte: Autor.

A Figura 14 apresenta o número total de práticas aplicadas por cada projeto da amostra sem *outliers*, é notável a predominância de valores entre 5 e 15, o que representa um baixo número de práticas, pois o PSMPO propõe 67 práticas.

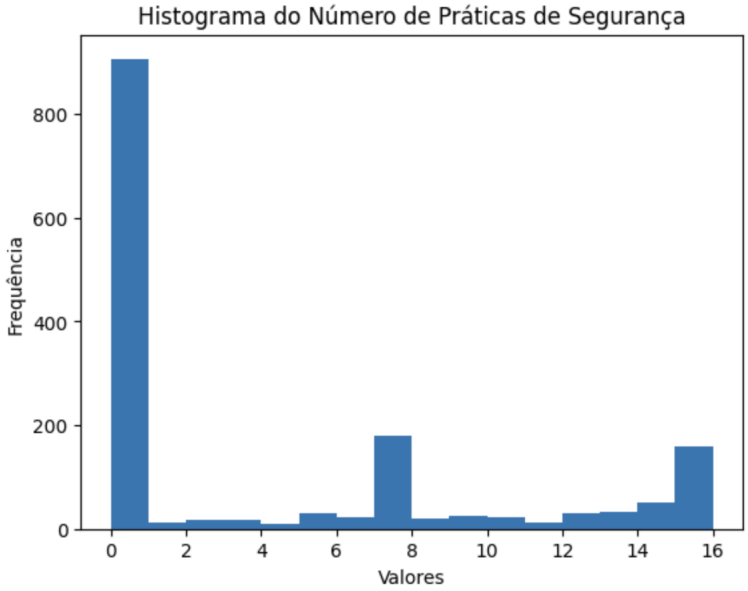
Figura 14 – Histograma do número total de práticas da amostra sem outliers de vulnerabilidades.



Fonte: Autor.

O PSMPO divide as práticas em conjuntos, um dos conjuntos é o de segurança, o qual aborda práticas relacionadas a criptografia, desenvolvimento seguro e solução de vulnerabilidades. Este conjunto possui 16 práticas, porém como é mostrado na Figura 15, mais de 50% dos projetos não aplicam nenhuma prática de segurança.

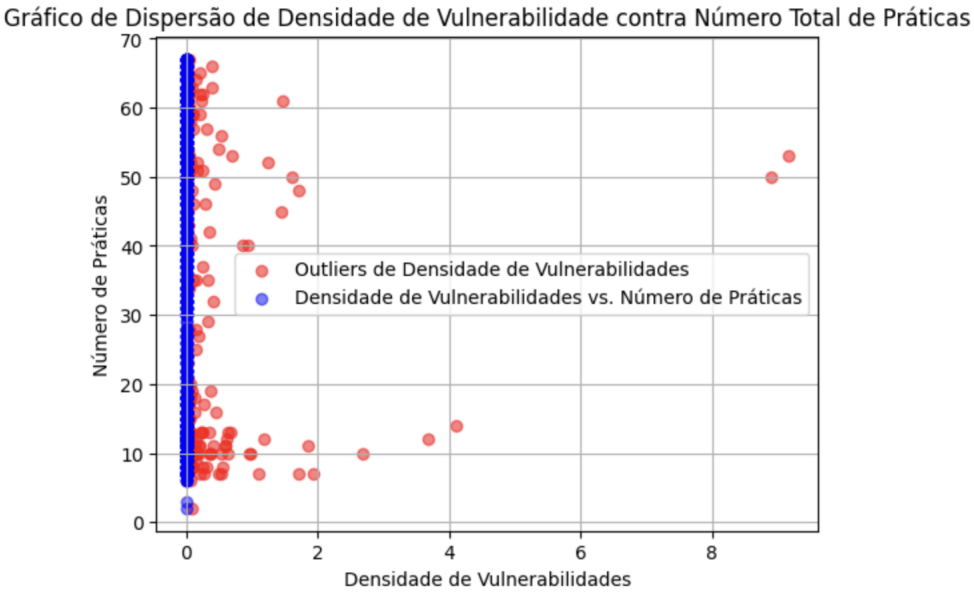
Figura 15 – Histograma do número de práticas de segurança da amostra sem outliers de vulnerabilidades.



Fonte: Autor.

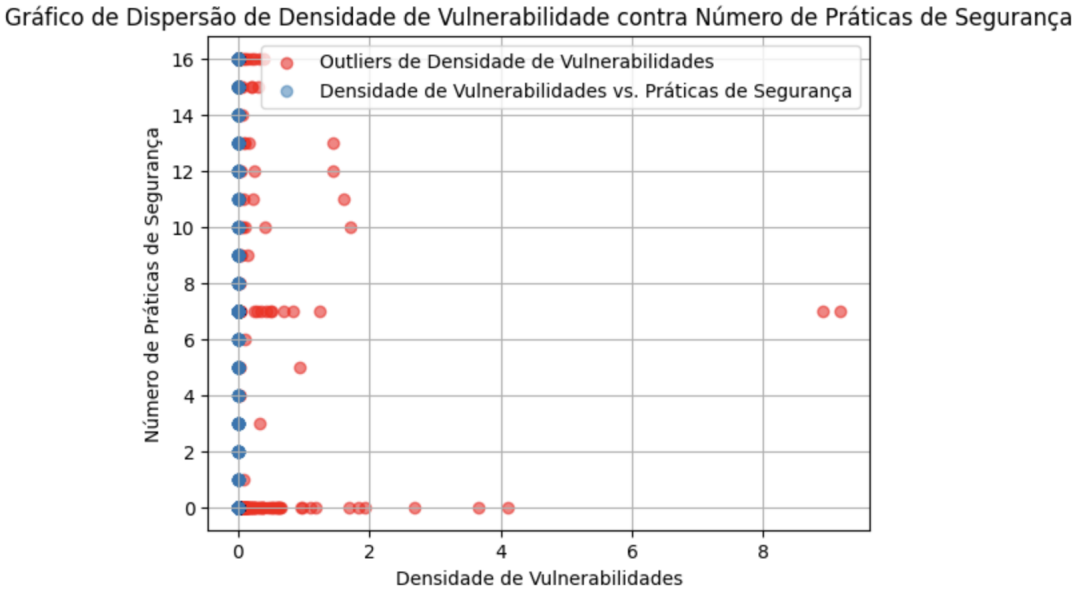
O gráfico de dispersão apresentado na Figura 16 possibilita a visualização de dispersão dos *outliers* e da amostra tratada, como todos os projetos não considerados *outliers* possuem densidade de vulnerabilidades igual a zero, é notável que não há uma correlação forte, pois a densidade segue em zero, independentemente do número total de práticas. O mesmo é notável em relação ao número de práticas de segurança como é apresentado na Figura 17.

Figura 16 – Gráfico de Dispersão de Densidade de vulnerabilidades contra número total de práticas.



Fonte: Autor.

Figura 17 – Gráfico de Dispersão de Densidade de vulnerabilidades contra número de práticas de segurança.



Fonte: Autor.

Para verificar o nível de correlação entre a densidade de vulnerabilidades e as práticas do PSMPO, foi utilizado o coeficiente de correlação de Spearman. A Tabela 7 apresenta o coeficiente calculado para o total de práticas e também para as categorias, as quais são subconjuntos das práticas, com a exclusão dos *outliers*.

A amostra utilizada para o cálculo do coeficiente foi a amostra com *outliers*, pois sem os *outliers* não há variação no valor da densidade de vulnerabilidade e portanto, não seria possível calcular o coeficiente.

Tabela 7 – Tabela de coeficiente de correlação de Spearman para conjuntos de práticas da amostra com outliers de densidade de vulnerabilidades.

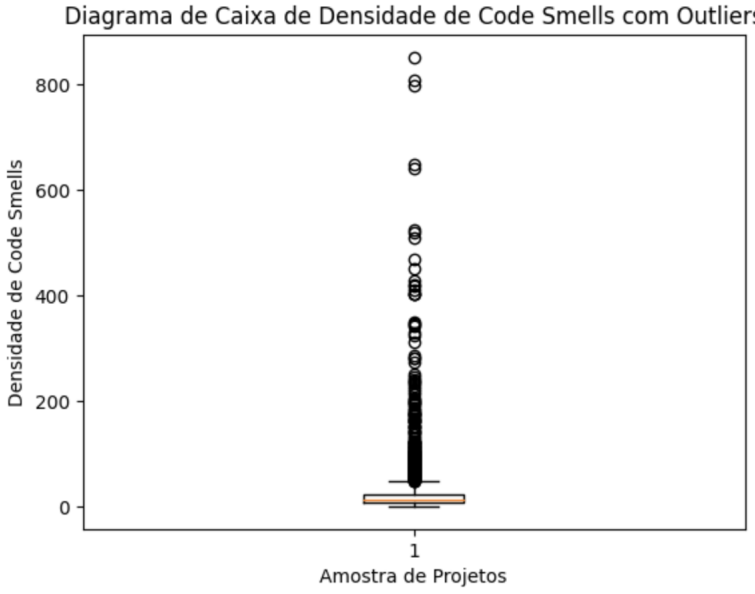
Categoria das Práticas	Coeficiente de Spearman
Geral	-0.07764
Básicas	-0.11690
Controle de Mudanças	0.01289
Reporting	-0.02184
Qualidade	-0.04989
Segurança	0.01182
Análise	-0.05476

Fonte: Autor.

5.3 Densidade de Code Smells

Para a visualização inicial dos dados da densidade de *code smells* foi utilizado um diagrama de caixa com a amostra de projetos. É notável a concentração dos quartis em valores menores que 100, e uma grande quantidade de dados anômalos, com uma grande variação, como é possível identificar no diagrama da Figura 18, em que 3 *outliers* chegam a ter o valor de densidade de *code smells* próximo ao 800.

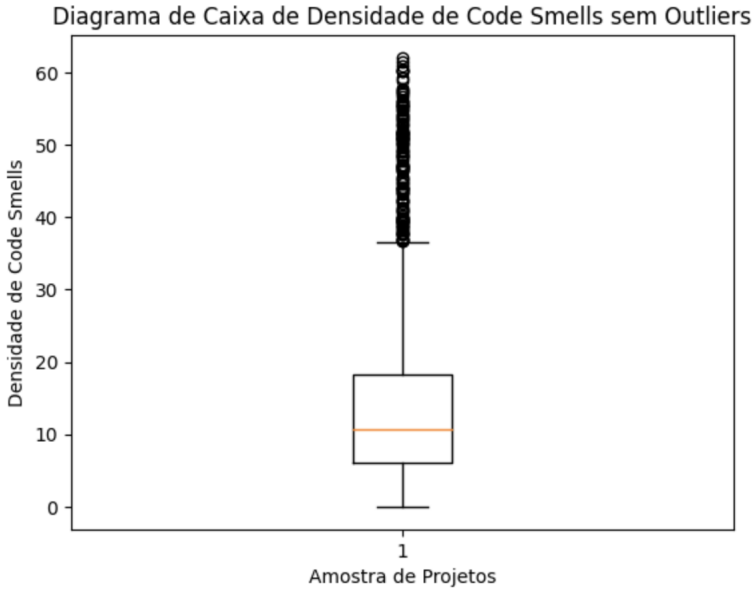
Figura 18 – Diagrama de Caixa de densidade de code smells com outliers.



Fonte: Autor.

Com o intuito de diminuir os ruídos dos dados e alcançar uma amostra de dados mais consistente, também foi aplicada a técnica de detecção de *outliers* de Tukey com o cálculo dos limites utilizando o *IQR*. O valor do parâmetro k utilizado foi 2,3 para exclusão mínima de valores.

Figura 19 – Diagrama de Caixa de densidade de code smells sem outliers.

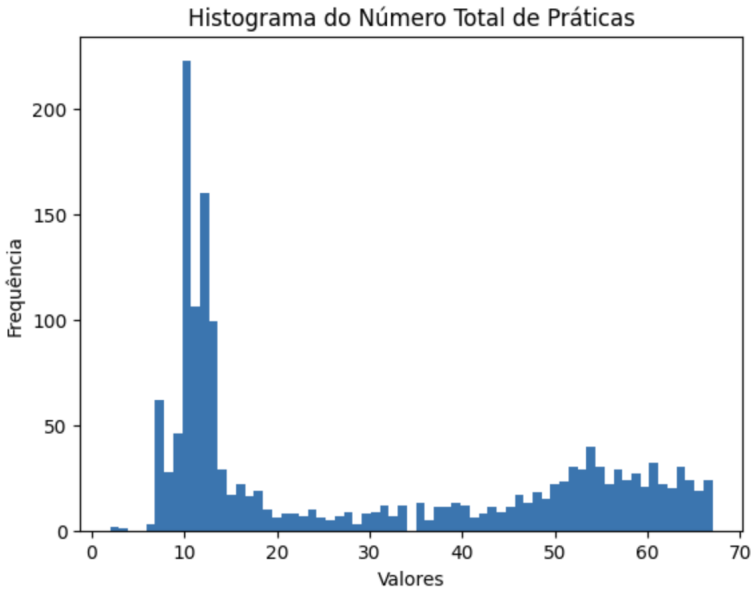


Fonte: Autor.

Após a remoção dos *outliers*, o diagrama de caixa do resultado apresentado na Figura 19 expõe de maneira mais clara os quartis, com os valores oscilando apenas até valores próximos de 60. A concentração está em valores menores, próximos de 10, como é indicado pela mediana no diagrama.

Com a base de informações sobre a densidade de *code smells* da amostra, é necessário compreender a variação e distribuição do número total de práticas do PSMPO. O histograma da Figura 20, apresenta uma grande aglomeração de projetos que aplicam entre 5 e 20 práticas. O que representa um valor baixo, pois apenas o subconjunto de práticas de segurança apresenta 16 práticas.

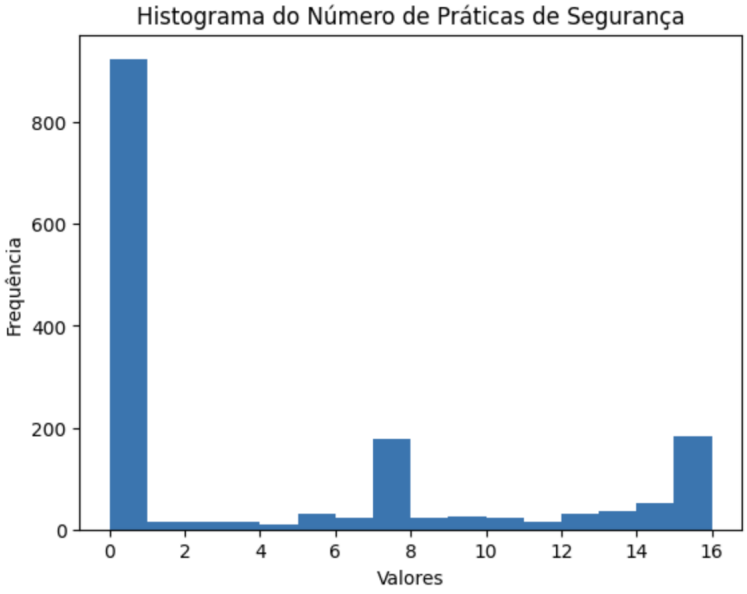
Figura 20 – Histograma do número total de práticas da amostra sem outliers de code smells.



Fonte: Autor.

O histograma da Figura 21 apresenta o número de práticas de seguranças dos projetos, mais de 50% dos projetos não aplicam nenhuma prática de segurança, aproximadamente 11% aplica 7 práticas e menos de 8% aplica as 16 práticas.

Figura 21 – Histograma do número de práticas de segurança da amostra sem outliers de code smells.

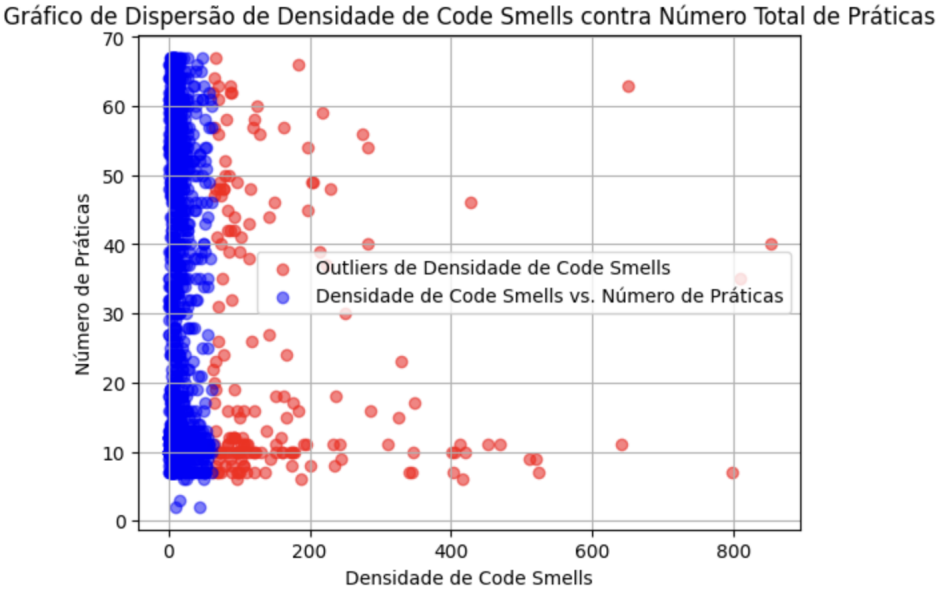


Fonte: Autor.

A Figura 22, um gráfico de dispersão da densidade de *code smells* contra o número total de práticas, apresenta uma variabilidade diferente da esperada, pois não é visível

uma correlação entre a densidade de *code smells* e o número total de práticas. A correlação não é notável entre os *outliers* e também não é notável entre a amostra após a remoção dos *outliers*.

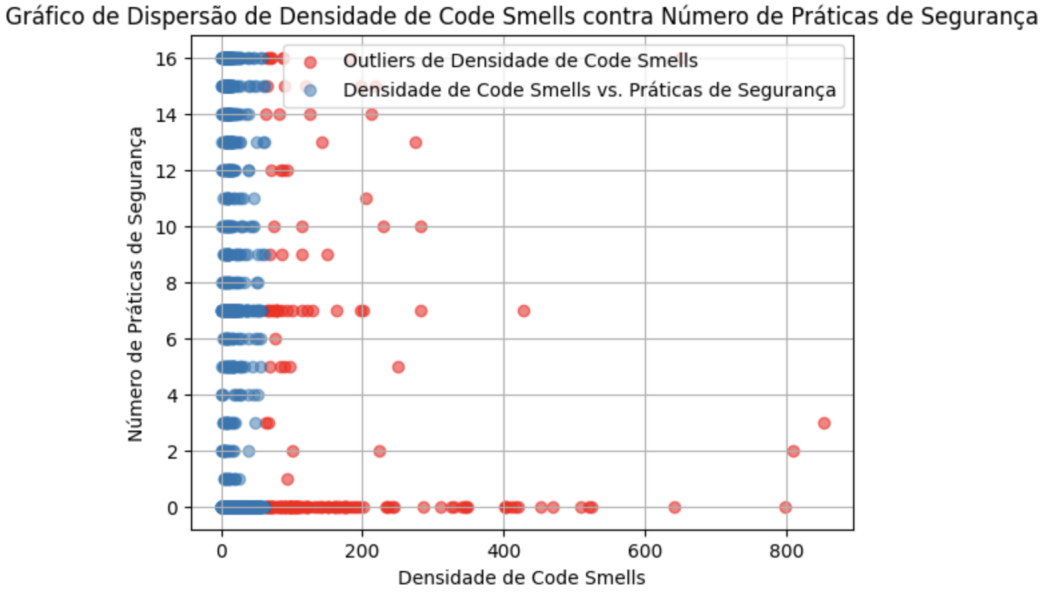
Figura 22 – Gráfico de Dispersão de Densidade de code smells contra número total de práticas.



Fonte: Autor.

A dispersão dos dados da Figura 23, também não apresenta a variação esperada, pois não há aparente correlação, seja positiva ou negativa. A densidade varia pouco, assim como o número de práticas, e com a pequena variação, não é visível uma correlação.

Figura 23 – Gráfico de Dispersão de Densidade de code smells contra número de práticas de segurança.



Fonte: Autor.

Com o intuito de avaliar a correlação entre a densidade de *code smells* e as práticas do PSMPO, recorreu-se ao emprego do coeficiente de correlação de Spearman. Na Tabela 8, encontra-se o referido coeficiente calculado tanto para o conjunto geral de práticas quanto para suas respectivas categorias, as quais configuram subconjuntos das práticas, com a eliminação dos valores anômalos.

Tabela 8 – Tabela de coeficiente de correlação de Spearman para conjuntos de práticas da amostra sen outliers de densidade de code smells.

Categoria das Práticas	Coeficiente de Spearman
Geral	-0.04734
Básicas	-0.00878
Controle de Mudanças	-0.03282
Reporting	-0.02295
Qualidade	-0.05516
Segurança	-0.01309
Análise	-0.03386

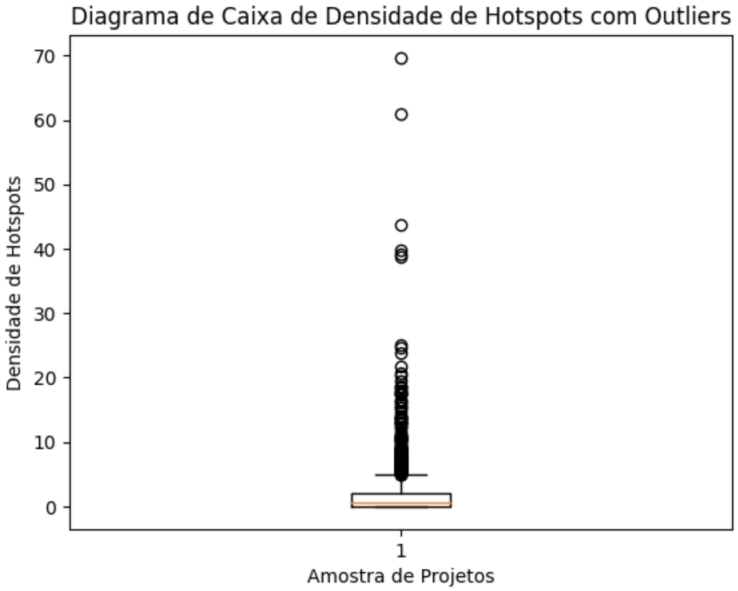
Fonte: Autor.

5.4 Densidade de Hotspots

A distribuição da densidade de *hotspots* está apresentada em forma de diagrama de caixa na Figura 24. É notável a concentração da distribuição em valores menores que 10 e uma mediana próxima ao 0. Alguns dados anômalos são perceptíveis, com a densidade

maior que 40, para remover esses casos extremos, foi aplicada a mesma técnica utilizada nos passos anteriores.

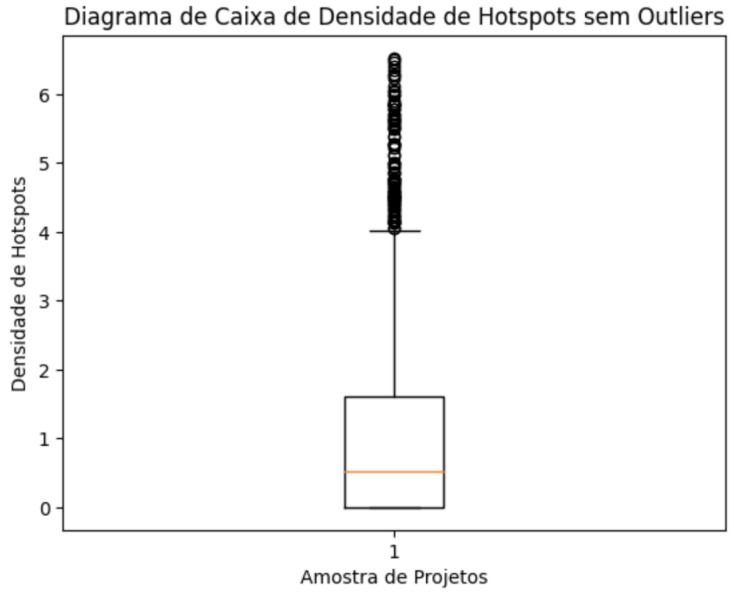
Figura 24 – Diagrama de Caixa de densidade de hotspots com outliers.



Fonte: Autor.

Sem os *outliers* detectados pela técnica de Tukey, como está apresentado na Figura 25, os valores se limitam a valores máximos próximos de 6. A mediana permanece próxima ao 0, a qual é um indicador de grande quantidade de projetos com densidades próximas ao 0.

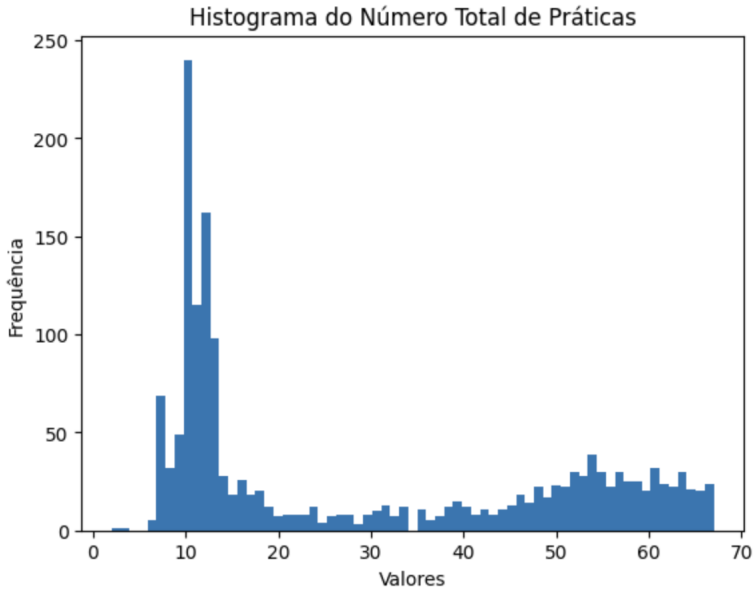
Figura 25 – Diagrama de Caixa de densidade de hotspots sem outliers.



Fonte: Autor.

O histograma da Figura 26 apresenta concentração de projetos com um baixo número total de práticas aplicadas, pois aplicam em sua maioria entre 5 e 15 práticas, o que representa menos de 25% das práticas propostas pelo PSMPO.

Figura 26 – Histograma do número total de práticas da amostra sem outliers de hotspots.



Fonte: Autor.

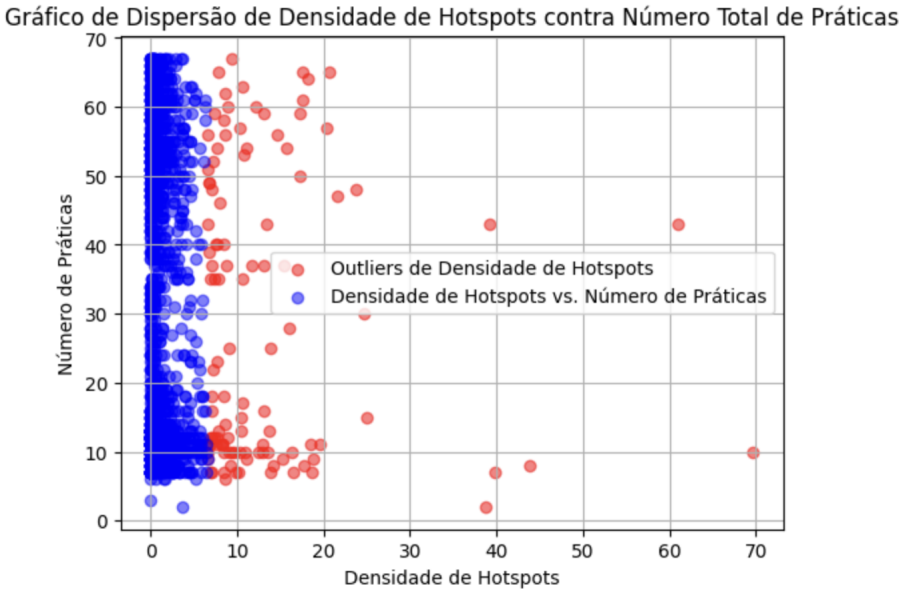
De acordo com o histograma da Figura 27, mais de 50% dos projetos não aplicam práticas de segurança, porém existem 2 aglomerações de projetos, com 7 e com 16 práticas, de maneira similar ao que é perceptível na amostra tratada de densidade de *code smells*.

Figura 27 – Histograma do número de práticas de segurança da amostra sem outliers de hotspots.

Fonte: Autor.

A variação da distribuição no gráfico de dispersão de densidade de *hotspots* contra o número total de práticas, na Figura 28, é maior do que a apresentada na densidade de *code smells*, porém também não é perceptível uma correlação entre as variáveis.

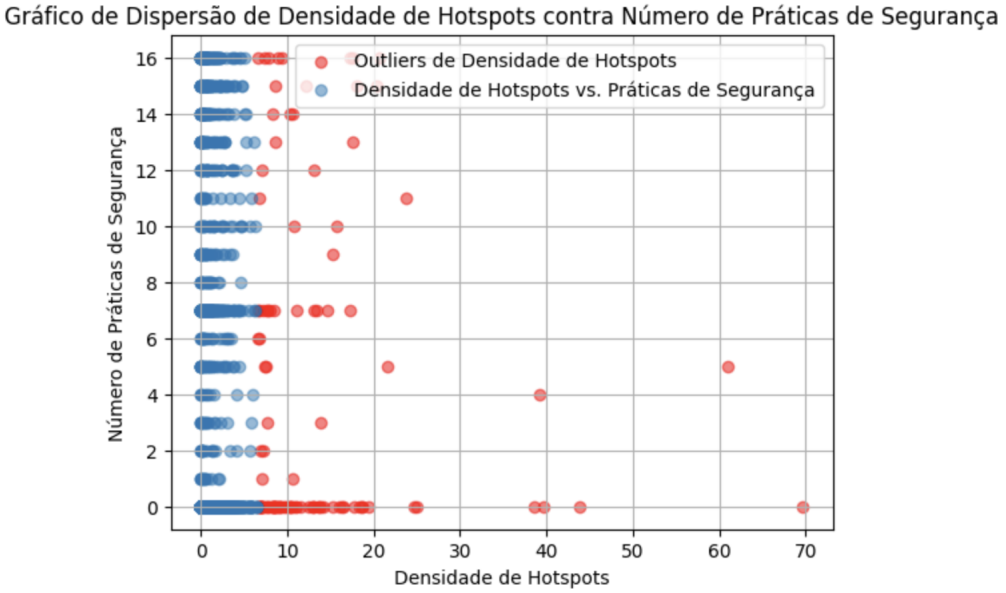
Figura 28 – Gráfico de dispersão de densidade de hotspots em função do número de práticas aplicadas por projeto.



Fonte: Autor.

A dispersão da densidade de *hotspots* em função do número de práticas de segurança apresentada na Figura 29, não apresenta uma visível correlação. Nem mesmo com a presença dos *outliers* é possível identificar um correlação, seja positiva ou negativa. Porém, a densidade de *hotspots* se mantém baixa independente do número de práticas aplicadas.

Figura 29 – Gráfico de dispersão de densidade de hotspots em função do número de práticas de segurança aplicadas por projeto.



Fonte: Autor.

Com o propósito de avaliar a correlação entre a densidade de *hotspots* e as práticas do PSMPO, foi calculado o coeficiente de correlação de Spearman. O coeficiente resultante para o conjunto completo de práticas, assim como para suas respectivas categorias é apresentado na Tabela 9. A amostra utilizada para o cálculo foi a amostra de projetos sem *outliers* da densidade de *hotspots*, pois é o conjunto de dados mais relevante.

Tabela 9 – Tabela de coeficiente de correlação de Spearman para conjuntos de práticas da amostra sem outliers de densidade de hotspots.

Categoria das Práticas	Coeficiente de Spearman
Geral	0.00971
Básicas	0.02465
Controle de Mudanças	0.01522
Reporting	0.02063
Qualidade	-0.02228
Segurança	0.02933
Análise	0.00563

Fonte: Autor.

5.4.1 Resultados

Os gráficos, diagramas e histogramas foram analisados em conjunto com as tabelas de coeficientes de Spearman. Em uma análise abrangente, não foi observada qualquer correlação evidente nos gráficos de dispersão para todas as densidades avaliadas. Os valores

apresentados nas tabelas indicam a ausência de correlação ou uma correlação extremamente fraca, uma vez que todos os coeficientes estão abaixo de 0,15 ou acima de -0,15.

Importante ressaltar que o coeficiente de Spearman varia de 1 a -1; quanto mais próximo dos extremos, maior a intensidade da correlação, enquanto valores próximos de zero indicam uma correlação mais fraca ou a ausência dela. Neste contexto, a presença de valores inferiores a 0,15 ou superiores a -0,15 sugere uma correlação fraca ou inexistente em todas as densidades analisadas.

6 Conclusão e Trabalhos Futuros

Neste trabalho, investigou-se o impacto da aplicação das melhores práticas do PSMPO na densidade de vulnerabilidades, com isso, foi estabelecido o objetivo geral de identificar se há uma relação entre o emprego de práticas do processo PSMPO no desenvolvimento de softwares de código aberto e seu nível de segurança indicado pelas densidades de vulnerabilidades, de *code smells* e de *hotspots* de segurança.

Para a compreensão dessa relação, estabeleceram-se três objetivos específicos. O primeiro, de identificar a densidade de vulnerabilidades, a densidade de *code smells* e a densidade de *hotspots* de segurança dos OSSs selecionados da base PSMPO, demandou o estudo sobre análise estática e sobre os conceitos de vulnerabilidades, *code smells* e *hotspots* de segurança.

Após a coleta das densidades definidas, foram utilizadas técnicas de *web scraping* para identificar as práticas do PSMPO empregadas em cada um dos OSSs selecionados. Então, com as densidades e as práticas coletadas, foi feita uma análise exploratória de dados para calcular coeficiente de correlação entre a densidade de vulnerabilidades, a densidade de *code smells* e a densidade de *hotspots* de segurança em relação ao número de práticas do PSMPO.

Além do cálculo, também foram analisados gráficos de dispersão para verificação de que há uma correlação visível. Também foram utilizados histogramas e diagramas de caixa, e com a análise dos dados, mostrou-se necessário o uso da técnica de detecção de *outliers* de Tukey para tornar os dados mais confiáveis e possibilitar uma melhor análise.

A aplicação da análise exploratória de dados, possibilitou a verificação de que a correlação entre a densidade de vulnerabilidades e o número de práticas do PSMPO é muito fraca, assim como as correlações das densidades de *code smells* e de *hotspots*. Este resultado não era o esperado, pois de acordo com a estrutura definição de níveis de completude das práticas do selo, o esperado era uma correlação forte e negativa. Também era esperado um número maior de práticas de segurança aplicadas, porém mais de 60% dos projetos não aplica nenhuma.

Em trabalhos futuros, pode-se comparar as densidades de vulnerabilidades, de *code smells* e de *hotspots* de segurança de projetos que aplicam as melhores práticas do PSMPO com OSSs que não aplicam. Esse estudo poderia trazer mais informações sobre a efetividade e relevância das práticas propostas. Além da possibilidade de identificar práticas que não estão presentes no PSMPO que podem aumentar a sua efetividade em melhorar a segurança dos seus projetos.

Referências

ALMARZOUQ, M. et al. Open Source: Concepts, Benefits, and Challenges. *Communications of the Association for Information Systems*, v. 16, 2005. ISSN 15293181. Disponível em: <https://aisel.aisnet.org/cais/vol16/iss1/37>. Citado na página 22.

BRETTHAUER, D. Open Source Software: A History. *Published Works*, dez. 2001. Disponível em: <https://opencommons.uconn.edu/libr_pubs/7>. Citado na página 10.

BRUCE, A.; BRUCE, P. *Estatística Prática para Cientistas de Dados*. [S.l.]: Alta Books, 2019. Google-Books-ID: b0mvDwAAQBAJ. ISBN 978-85-508-1080-5. Citado na página 25.

Common Weakness Enumeration. *CWE - 2022 CWE Top 25 Most Dangerous Software Weaknesses*. 2022. Disponível em: <https://cwe.mitre.org/top25/archive/2022/2022_cwe_top25.html>. Citado 2 vezes nas páginas 14 e 15.

CROWSTON, K.; HOWISON, J. The social structure of free and open source software development. *First Monday*, fev. 2005. ISSN 1396-0466. Disponível em: <https://firstmonday.org/ojs/index.php/fm/article/view/1207>. Citado na página 22.

Free Software Foundation. *What is Free Software? - GNU Project - Free Software Foundation*. 2019. Disponível em: <https://www.gnu.org/philosophy/free-sw.html>. Citado 3 vezes nas páginas 10, 20 e 21.

GEER, D. Are Companies Actually Using Secure Development Life Cycles? *Computer*, v. 43, n. 6, p. 12–16, jun. 2010. ISSN 1558-0814. Conference Name: Computer. Disponível em: <https://ieeexplore-ieee-org.ez54.periodicos.capes.gov.br/document/5481927>. Citado 2 vezes nas páginas 15 e 16.

GIL, A. C. *Como elaborar projetos de pesquisa*. 4. ed.. ed. [S.l.]: Editora Atlas S.A., 2002. ISBN 85 · 224-3169-8. Citado 2 vezes nas páginas 12 e 27.

GitHub. *Adding a workflow status badge*. 2023. Disponível em: <https://ghdocs-prod.azurewebsites.net/en/actions/monitoring-and-troubleshooting-workflows/adding-a-workflow-status-badge>. Citado na página 23.

GLEZ-PEñA, D. et al. Web scraping technologies in an API world. *Briefings in Bioinformatics*, v. 15, n. 5, p. 788–797, set. 2014. ISSN 1467-5463. Disponível em: <https://doi.org/10.1093/bib/bbt026>. Citado na página 28.

HOAGLIN, D. C.; IGLEWICZ, B. Fine-Tuning Some Resistant Rules for Outlier Labeling. *Journal of the American Statistical Association*, v. 82, n. 400, p. 1147–1149, 1987. ISSN 0162-1459. Publisher: [American Statistical Association, Taylor & Francis, Ltd.]. Disponível em: <https://www.jstor.org/stable/2289392>. Citado na página 39.

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 14 e 15.

MCGRAW, G. *Software Security: Building Security In*. [s.n.], 2006. ISBN 978-0-321-35670-3. Disponível em: <https://learning.oreilly.com/library/view/software-security-building/0321356705/>. Citado na página 15.

Microsoft. *SDL Process Guidance Version 5.2*. 2012. Disponível em: <https://www.microsoft.com/en-us/download/details.aspx?id=29884>. Citado 2 vezes nas páginas 16 e 24.

MOORE, D. S. *Introduction to the Practice of Statistics*. WH Freeman and company, 2009. Disponível em: <https://ds.amu.edu.et/xmlui/bitstream/handle/123456789/14700/Introduction%20to%20the%20Practice%20of%20Statistics%20%20-%201010%20pages.pdf?sequence=1&isAllowed=y>. Citado na página 25.

Open Source Iniciative. *The Open Source Definition*. 2006. Disponível em: <https://opensource.org/osd/>. Citado 3 vezes nas páginas 10, 19 e 21.

Open Source Iniciative. *Licenses*. 2022. Disponível em: <https://opensource.org/licenses/>. Citado na página 21.

Open Source Security Foundation. *FLOSS Best Practices Criteria (All Levels)*. 2021. Disponível em: <https://bestpractices.coreinfrastructure.org/en/criteria>. Citado 5 vezes nas páginas 11, 24, 28, 36 e 37.

Open Source Security Foundation. *OpenSSF Best Practices Badge Program*. 2021. Disponível em: <https://bestpractices.coreinfrastructure.org/en>. Citado na página 22.

Open Source Security Foundation. *About OpenSSF*. 2023. Disponível em: <https://openssf.org/about/>. Citado 3 vezes nas páginas 22, 23 e 24.

OpenLogic by Perforce. *2023 State of Open Source Report*. 2023. Disponível em: <https://www.openlogic.com/success/2023-state-open-source-report>. Citado na página 10.

PETERSON, K. The GitHub Open Source Development Process. dez. 2013. Disponível em: <https://www.researchgate.net/profile/Kevin-Peterson-8/publication/259217367_The_GitHub_Open_Source_Development_Process/links/02e7e52a762dcce47b000000/The-GitHub-Open-Source-Development-Process.pdf>. Citado na página 21.

PIESSENS, F. The Cyber Security Body of Knowledge v1.1.0, 2021. In: . University of Bristol, 2021. Section: Software Security. Disponível em: <https://www.cybok.org/>. Citado na página 14.

RANSOME, J.; MISRA, A. *Core Software Security: Security at the Source*. London, UNITED STATES: Auerbach Publishers, Incorporated, 2013. ISBN 978-1-4665-6096-3. Disponível em: <http://ebookcentral.proquest.com/lib/univbrasilia-ebooks/detail.action?docID=1547083>. Citado 3 vezes nas páginas 14, 15 e 17.

SINGH, S.; KAUR, S. A systematic literature review: Refactoring for disclosing code smells in object oriented software. *Ain Shams Engineering Journal*, v. 9, n. 4, p. 2129–2151, dez. 2018. ISSN 2090-4479. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2090447917300412>. Citado na página 15.

SonarSource S.A. *Languages overview*. 2023. Disponível em: <https://docs.sonarsource.com/sonarqube/10.2/analyzing-source-code/languages/overview/>. Citado na página 28.

SonarSource S.A. *An overview of the key concepts used within SonarQube*. 2023. Disponível em: <https://docs.sonarsource.com/sonarqube/10.2/user-guide/concepts/>. Citado na página 15.

Synopsys, Inc. *2023 OSSRA Report*. 2023. Disponível em: <https://www.synopsys.com/software-integrity/engage/ossra/rep-ossra-2023-pdf>. Citado na página 10.

SáNCHEZ, M. C. et al. Software vulnerabilities overview: A descriptive study. *Tsinghua Science and Technology*, v. 25, n. 2, p. 270–280, abr. 2020. ISSN 1007-0214. Conference Name: Tsinghua Science and Technology. Citado na página 11.

The Linux Foundation. *Why CII best practices gold badges are important - Linux Foundation*. 2020. Disponível em: <https://www.linuxfoundation.org/blog/blog/why-cii-best-practices-gold-badges-are-important>. Citado na página 24.

The OWASP Foundation. *OWASP SAMM Version 2*. 2020. Disponível em: <https://drive.google.com/file/u/0/d/1cI3Qzfrly_X89z7StLWI5p_Jfqs0-OZv/view?pli=1&usp=embed_facebook>. Citado 3 vezes nas páginas 18, 19 e 24.

The OWASP Foundation. *OWASP Top Ten | OWASP Foundation*. 2021. Disponível em: <https://owasp.org/www-project-top-ten/>. Citado na página 14.

THUNG, F. et al. Network Structure of Social Coding in GitHub. In: *2013 17th European Conference on Software Maintenance and Reengineering*. [s.n.], 2013. p. 323–326. ISSN: 1534-5351. Disponível em: <https://ieeexplore.ieee.org/document/6498480>. Citado na página 21.

TUKEY, J. W. *Exploratory data analysis as part of a larger whole*. Proceedings of the 18th conference on design of experiments in Army research and development I. Washington, DC: [s.n.], 1972. v. 1010. Disponível em: <https://apps.dtic.mil/sti/pdfs/AD0776910.pdf#page=18>. Citado na página 25.

VIEGA, J.; MCGRAW, G. *Building Secure Software*. [s.n.], 2002. ISBN 978-0-672-33409-2. Disponível em: <https://learning.oreilly.com/library/view/building-secure-software/9780672334092/fm.html>. Citado 2 vezes nas páginas 14 e 15.

WANSTRATH, C. *We Launched*. 2008. Disponível em: <https://github.blog/2008-04-10-we-launched/>. Citado na página 21.