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

Quais as dúvidas dos desenvolvedores sobre segurança de software? Uma análise a partir das perguntas e respostas no StackExchange

Autores: Emily Dias Sousa e Vitor Magalhães Lamego
Orientadora: Profa. Dra. Elaine Venson

Brasília, DF
2023

Emily Dias Sousa e Vitor Magalhães Lamego

Quais as dúvidas dos desenvolvedores sobre segurança de software? Uma análise a partir das perguntas e respostas no StackExchange

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. Elaine Venson

Brasília, DF

2023

Agradecimentos

Em primeiro lugar, agradecemos a Deus por ter nos permitido chegar onde chegamos e por ter nos guiado até aqui.

Agradecemos à Profa. Dra. Elaine pela sua dedicação e comprometimento em ser nossa orientadora do TCC. Somos gratos pela sua paciência, disponibilidade e por compartilhar seus conhecimentos e experiências com a gente, isso nos ajudou bastante.

Gostaríamos de agradecer aos nossos amigos e a nossa família por todo o apoio durante os momentos mais difíceis e por ter apoiado a gente ao longo de nossa jornada. Muito obrigado!

"Com grandes poderes vêm grandes responsabilidades"
(Stan Lee)

Resumo

Com o aumento do desenvolvimento de software e o surgimento de ataques virtuais, é de extrema importância que os desenvolvedores estejam atentos à segurança para garantir a privacidade e integridade dos dados dos usuários. Por segurança de software ser um tema abrangente, é necessário investigar tópicos e tendências relevantes na área de segurança, a fim de fornecer um guia útil para pesquisadores e profissionais do campo. Com isto, este estudo busca responder a seguinte pergunta: "Quais as dúvidas dos desenvolvedores sobre segurança de software?". Para tanto, este trabalho apresenta um estudo sobre questões relacionadas à segurança de software em sites de perguntas e respostas do *Stack Exchange Q&A*, que são fóruns online onde os desenvolvedores trocam informações com outras pessoas sobre suas dificuldades. Foram utilizados sites como o *Stack Overflow*, um dos fóruns mais populares entre os desenvolvedores, o *IT Security*, por ser mais focado em questões de segurança e, por fim, o *Software Engineering*, um fórum mais focado em engenharia de software, que é utilizado por profissionais, acadêmicos e estudantes que trabalham no ciclo de vida do desenvolvimento de sistemas. O principal objetivo deste trabalho foi identificar as áreas de maior dificuldade e interesse dos desenvolvedores. Para alcançar esse objetivo, a metodologia adotada incluiu pesquisa exploratória no *Stack Exchange* aplicando o algoritmo LDA (*Latent Dirichlet Allocation*) para extrair os tópicos da base de dados e realizar a análise das categorias das perguntas e as *tags* utilizadas. Os resultados encontrados foram cerca de 16 tópicos com 16 *tags* gerais. Para obter esses valores foi utilizado cerca de 104.921 perguntas dos fóruns. Este estudo mostra que as perguntas demoram dias para serem respondidas, com uma grande quantidade de respostas, porém um percentual baixo para respostas aceitas, com isto, pode-se concluir que a comunidade tem diversas questões e dificuldades sobre o assunto, e pelo fato de não ter questões muito definidas pode mostrar que a comunidade dos fóruns não dominam muito sobre o assunto.

Palavras-chave: Segurança de Software; *Stack Exchange Q&A*; Desenvolvimento de Software Seguro; *Stack Overflow*; LDA.

Abstract

With the increasing development of software and the emergence of virtual attacks, it is of utmost importance for developers to be vigilant about security to ensure the privacy and integrity of user data. As software security is a comprehensive theme, it is necessary to investigate relevant topics and trends in the security field to provide a useful guide for researchers and professionals in the field. Therefore, this study aims to address the following question: "What are the doubts of developers regarding software security?". To achieve this, the research presents a study on issues related to software security on question and answer sites on Stack Exchange Q&A, which are online forums where developers exchange information with others about their difficulties. Sites such as Stack Overflow, one of the most popular forums among developers, IT Security, focused on security issues, and Software Engineering, a forum more focused on software engineering used by professionals, academics, and students working in the software development life cycle, were utilized. The main objective of this work was to identify the areas of greatest difficulty and interest for developers. To achieve this objective, the adopted methodology included exploratory research on Stack Exchange, applying the Latent Dirichlet Allocation (LDA) algorithm to extract topics from the database and analyze the categories of questions and the tags used. The results revealed approximately 16 topics with 16 general tags, based on around 104,921 forum questions. This study indicates that questions take days to be answered, with a large number of responses but a low percentage of accepted answers. Consequently, it can be inferred that the community has various questions and difficulties on the subject. The lack of well-defined questions may suggest that the forum community does not have extensive expertise in the subject.

Key-words: Software Security; Stack Exchange Q&A; Secure Software Development; Stack Overflow; LDA.

Lista de abreviaturas e siglas

SO	Stack Overflow
SE	Software Engineering
SSDLC	Secure Software Development Life Cycle
SDL	Security Development Lifecycle
SAMM	Software Assurance Maturity Model
BSIMM	Building Security In Maturity Model
API	Application Programming Interface
SDLC	Software Development Life Cycle
LDA	Latent Dirichlet Allocation
LSA	Latent Semantic Analysis
pLSA	Probabilistic Latent Semantic Analysis
ITSec	IT Security

Lista de símbolos

K	número de tópicos.
n	número de palavras do vocabulário.
m	número de documentos.
n_{d_j}	número de palavras em um documento d_j, onde $1 \leq j \leq m$.
θ	distribuição de tópicos por documentos.
ϕ	distribuição dos tópicos sobre as palavras do vocabulário.
θ_j	vetor com a proporção dos tópicos para o documento d_j, onde $1 \leq j \leq m$.
ϕ_k	vetor com a proporção das palavras do vocabulário para o tópico k, onde $1 \leq k \leq K$.
α	relacionada à distribuição documento-termo.
β	relacionada à distribuição tópico-palavra.
w_i	i-ésima palavra do vocabulário, onde $1 \leq i \leq n$.
$w_{j,i}$	palavra w_i observada no documento d_j, onde $1 \leq j \leq m$ e $1 \leq i \leq n$.
$z_{j,i}$	distribuição de tópicos associada à palavra $w_{j,i}$ no documento d_j, onde $1 \leq j \leq m$ e $1 \leq i \leq n$.

Sumário

1 INTRODUÇÃO . . . 15
1.1 Problema . . . 16
1.2 Objetivo . . . 16
1.3 Metodologia . . . 17
1.4 Organização do Documento . . . 18

2 REFERENCIAL TEÓRICO . . . 20
2.1 Segurança de Software . . . 20
2.2 Vulnerabilidades . . . 21
2.2.1 Categoria de Vulnerabilidades . . . 21
2.2.1.1 Vulnerabilidades no Gerenciamento de Memória . . . 21
2.2.1.2 Vulnerabilidades de Injeção . . . 22
2.2.1.3 Vulnerabilidades de Condição de Corrida . . . 22
2.2.1.4 Vulnerabilidades de API . . . 23
2.2.2 Prevenção . . . 23
2.2.3 Detecção . . . 24
2.3 Processos de ciclo de vida de software seguro . . . 25
2.3.1 Ciclo de vida de desenvolvimento de software seguro (SSDLC) . . . 26
2.3.2 Construindo segurança no modelo de maturidade (BSIMM) . . . 27
2.3.3 Modelo de Maturidade de Garantia de Software (SAMM) . . . 29
2.3.4 Ciclo de vida de desenvolvimento de segurança (SDL) . . . 31
2.4 Stack Overflow e Fóruns de Perguntas e Respostas . . . 35
2.5 Alocação de Dirichlet Latente (LDA) . . . 36

3 METODOLOGIA . . . 39
3.1 Plano metodológico . . . 39
3.2 Planejamento da Pesquisa . . . 40
3.3 Referências Bibliográficas . . . 40
3.4 Plano de ação . . . 42
3.4.1 Extração das perguntas e respostas . . . 42
3.4.2 Categorização das perguntas com LDA . . . 43
3.4.3 Análise das *Tags* relacionadas com as perguntas . . . 44
3.4.4 Identificação das perguntas com categorias mais difíceis . . . 44

4 RESULTADOS . . . 46
4.1 Extração das perguntas e respostas . . . 46

4.2 Categorização das perguntas com LDA 47
4.2.1 Análise Exploratória 47
4.2.2 Carregando dados 49
4.2.2.1 Divisão dos dados 49
4.2.2.2 Escolha dos dados 50
4.2.2.3 Categorização dos Tópicos 51
4.2.3 Resultados 52
4.3 Análise das Tags relacionadas às perguntas 55
4.3.1 Resultados 56
4.3.1.1 Tags mais utilizadas geral 56
4.3.1.2 Tags mais utilizadas de cada fórum 57
4.4 Identificação das perguntas mais difíceis 61
4.4.1 Implicações 62
4.5 Comparação dos Estudos 64

5 CONCLUSÃO 67

REFERÊNCIAS 69

APÊNDICES 73

APÊNDICE A – CÓDIGO SQL 74

APÊNDICE B – GRÁFICO DE CATEGORIAS POR ANO 76

1 Introdução

Segurança de Software é uma área de estudo dentro de engenharia de software responsável por garantir que uma determinada aplicação funcione de maneira correta perante ataques maliciosos (MCGRAW, 2006). A crescente integração de software em diversos aspectos da vida realça a importância da segurança como um requisito essencial para qualquer programa (MOHAMMED et al., 2017).

A segurança de software está apoiada em três pilares que são essenciais para que um determinado produto seja classificado como seguro: confidencialidade, integridade e disponibilidade. Esses três aspectos precisam ser considerados durante todo o ciclo de desenvolvimento para garantir a segurança do produto final(RANSOME; MISRA, 2018).

Apesar da importância destacada para a área, é muito comum que a segurança de um software seja trabalhada apenas nas fases finais do ciclo de desenvolvimento, o que influencia em um grande risco de vulnerabilidades introduzidas no produto final (UMAIR; ZULKERNINE, 2009). Isso se deve pelo fato de ser tratado mais como uma decisão de negócio do que uma necessidade ou requisito do sistema (RANSOME; MISRA, 2018). Essa ambiguidade ou inconsistência no planejamento do projeto ocorre pelo fato de existir uma separação clara entre os requisitos para a segurança do software e os requisitos funcionais do sistema, por conta disso, é necessário um esforço para acoplar as rotinas, tarefas de segurança e as suas necessidades junto aos requisitos funcionais desde o princípio do projeto, evitando que futuramente a segurança seja tratada como uma decisão de negócio (KHAN; ZULKERNINE, 2009).

A importância do assunto e a falta de prioridade dentro do ciclo de desenvolvimento pode ser entendido pelo fato da área de segurança ser algo recente para o mundo do software. Os seus primeiros artigos e estudos foram publicados no ano de 2001, o que explica o porquê de ainda ser um assunto em que desenvolvedores, arquitetos e entusiastas da área não seguem as melhores práticas ou até mesmo nem adotam as ideias dentro do ciclo de desenvolvimento (MCGRAW, 2006).

Os desenvolvedores costumam utilizar a ferramenta de busca do Google (HORA, 2021) para pesquisar sobre suas dúvidas, questões ou curiosidades. Entre os resultados da pesquisa, geralmente são apresentados fóruns de perguntas e respostas. Há uma série de fóruns disponíveis para a comunidade, sendo o Stack Overflow[1] um exemplo de plataforma com uma das comunidades mais ativas, possuindo mais de cem milhões de acessos mensais.

Os fóruns online fornecem aprendizados e conexões de participantes em todo o mundo (REKHA; VENKATAPATHY, 2015) e favorecem a troca de experiências e conhe-

[1] https://stackoverflow.com/

cimentos. Neste contexto, programadores e entusiastas pesquisam sobre diversos temas, incluindo códigos (SADOWSKI; STOLEE; ELBAUM, 2015), metodologias, documentações, correções de *bugs* (HORA, 2021).

Nos últimos anos, o tema da proteção de dados pessoais tem ganhado destaque no contexto tecnológico. Isso ocorre devido à crescente dependência das pessoas em softwares no cotidiano, que frequentemente requerem acesso a dados, inclusive informações sensíveis, para proporcionar uma experiência satisfatória ou cumprir suas funcionalidades.

No entanto, ao lado das preocupações com vazamentos de dados, também se destaca a questão do uso indevido, como a venda de dados sem o consentimento e a falta de conscientização das partes envolvidas (RAPôSO et al., 2019). Sendo assim, o avanço tecnológico traz muitos benefícios, mas com ele vêm alguns pontos de extrema atenção, sendo a segurança de software um dos principais (HANIF et al., 2021).

1.1 Problema

Constatado o problema e a disparidade entre a importância da segurança de um software e a sua efetiva aplicação dentro dos projetos desenvolvidos, se faz necessário entender as principais questões e dúvidas que permeiam o meio de desenvolvimento de software, para que assim as informações sejam fornecidas de maneira mais efetiva, atacando os pontos de mais dificuldade dos desenvolvedores e profissionais da área.

Os sites classificados como *Stack Exchange Q&A*, ou também conhecidos como fóruns, abrigam vastos bancos de perguntas e respostas fornecidas por usuários ao longo dos anos e consequentemente um amplo conteúdo enriquecedor sobre diferentes assuntos. Entretanto, não é tão simples a navegação ou obtenção desses dados classificados por assunto, sendo muito comum a navegação por questões individualizadas. Isso pode ser um obstáculo para quem procura aprender mais sobre um determinado assunto de maneira mais geral e não tão individualizada como se encontra na maioria das questões.

Desse modo, a questão principal deste estudo é:

O que está sendo mais discutido sobre segurança de software em sites do tipo *Stack Exchange Q&A* e quais são as principais dificuldades dos profissionais da área?

1.2 Objetivo

Sabendo do potencial que os sites de *Stack Exchange Q&A*[2] possuem, e a quantidade de informações presentes nessas ferramentas, o estudo observacional em questão

[2] https://data.stackexchange.com/

tem como objetivo principal entender onde se encontram as principais lacunas de conhecimento na área de Segurança de Software, para que assim os educadores e profissionais da área consigam prestar um suporte mais direcionado para os assuntos de maior dificuldade.

Os três objetivos específicos são:

1. Compreender como os usuários de diferentes sites de perguntas e respostas categorizam as questões da área. Entender quais são as principais *tags* utilizadas por categoria criada no estudo, além de verificar as *tags* mais utilizadas no contexto geral de Segurança de Software.

2. Identificar as categorias de perguntas que apresentam mais dificuldades em obter respostas satisfatórias. Isso é importante para ajudar educadores e profissionais a direcionar recursos e atenção para superar esses desafios e aumentar o conhecimento e a compreensão da área.

3. Comparar os resultados desse estudo com base no artigo do Rahman (2016), pelo fato de ter o tema parecido, onde será possível ver quais dificuldades os desenvolvedores estão enfrentando sobre o tema ao decorrer do tempo.

1.3 Metodologia

A metodologia definida para este trabalho foi baseada em um artigo sobre um estudo observacional com o tema de dívida técnica (ALFAYEZ et al., 2023), cuja metodologia consiste em fazer uma pesquisa exploratória antes da realização da coleta de dados.

Este estudo será realizado utilizando três diferentes fóruns: *Stack Overflow*[3], por ser o ambiente mais conhecido pelos desenvolvedores, *IT Security*[4], por ser focado na área de segurança e por fim o *Software Engineering*[5]. Foi feito uma pesquisa geral no site de *Stack Exchange Q&A*[6] para encontrar fóruns além do SO que falasse sobre segurança de software, e foi encontrado o IT Security (ITSec), onde ele tem cerca de 68 mil questões, com isto, o fórum foi escolhido para incrementar em nossa pesquisa. Existe outro fórum sobre segurança, porém ele tem somente 1,1 mil perguntas, por este motivo, foi decidido escolher o ITSec. A escolha do *Software Engineering* se deve ao fato de ser um fórum que trata sobre metodologias de software e por ter cerca de 63 mil questões, um número que foi considerado bom para acrescentar na pesquisa. Ao explorar esses três fóruns - um abrangendo temas gerais, outro focado em metodologias e um terceiro dedicado à

[3] https://stackoverflow.com/
[4] https://security.stackexchange.com/
[5] https://softwareengineering.stackexchange.com/
[6] https://data.stackexchange.com/

segurança de software - percebemos que eles forneceriam as questões desejadas para a pesquisa.

Um ponto importante é a escolha por utilizar questões e strings de busca na língua inglesa. Após ser feita uma análise sobre a quantidade de dados disponíveis nos fóruns chegou-se a conclusão de que a língua inglesa oferece mais recursos para análises e resultados.

A partir do levantamento das perguntas e respostas que serão utilizadas, o trabalho consiste em responder as seguintes questões:

1. **Quais categorias relacionadas a segurança de software são abordadas nos sites de perguntas e respostas do *Stack Exchange*?**

   Realização de uma categorização automatizada como forma de apoio para as análises das etapas seguintes.

2. **Quais tags e categorias específicas são usadas em perguntas relacionadas à segurança de software em sites de perguntas e respostas do *Stack Exchange*?**

   Estudo sobre as *tags* utilizadas nas perguntas filtradas, a fim de elucidar como os usuários classificam as suas perguntas.

3. **Quais categorias relacionadas à segurança de software são as mais difíceis? E comparado aos modelos práticos da área ?**

   A partir dos dados de respostas obtidas e resposta aceita será realizado um levantamento para definir a dificuldade das questões em cada conjunto de categoria. Posteriormente será realizada uma comparação entre os resultados obtidos e os modelos práticos existentes na área.

A partir desses resultados, a comunidade poderá se beneficiar encontrando as informações relevantes para as suas necessidades sobre segurança de software, tornando mais fácil para os usuários encontrarem respostas para suas perguntas específicas.

1.4 Organização do Documento

O trabalho está organizado por capítulo, o primeiro deles é este em questão, que trabalhou uma introdução sobre os fóruns de pesquisa, segurança de software e o trabalho que foi desenvolvido. Em sequência a esse capítulo teremos os seguintes:

- **Capítulo 2 - Referencial Teórico:** capítulo responsável por informar toda a base de conhecimento necessário para que seja realizado o estudo, informando ao leitor as

pesquisas utilizadas, assim como os conhecimentos necessários para o entendimento do estudo.

- **Capítulo 3 - Metodologia:** capítulo responsável por explicar com mais detalhes a metodologia utilizada no estudo.
- **Capítulo 4 - Resultados:** capítulo responsável por mostrar como foi feito o passo a passo do estudo.
- **Capítulo 5 - Comparação dos Estudos:** capítulo que mostra a comparação dos resultados com estudos anteriores.
- **Capítulo 6 - Conclusão:** capítulo que mostra a conclusão do estudo.

2 Referencial Teórico

Este capítulo explica os principais conceitos teóricos que servirão de base para a pesquisa e desenvolvimento deste trabalho. Serão discutidos assuntos relevantes para a compreensão do contexto em que o estudo se encontra, trazendo conceitos teóricos fundamentais para embasar as análises críticas.

2.1 Segurança de Software

Existem três principais pilares que fundamentam a segurança de um software, sendo eles: confidencialidade, integridade e disponibilidade. Esses três aspectos devem ser garantidos para os dados e funcionalidades de um sistema (PIESSENS, 2021).

A confidencialidade está atrelada a restringir quem pode ter acesso a determinado dado ou funcionalidade de um sistema. A integridade significa que um software deve garantir que os dados não sejam alterados. Por exemplo, uma comunicação modificada dentro de um banco de dados, poderia influenciar significativamente o rumo de um julgamento. Por fim, a disponibilidade de um determinado sistema também está atrelada à segurança do software, uma vez que um sistema de segurança não disponível poderia permitir invasões e ataques de diferentes formas, trazendo prejuízos a um determinado usuário, por conta disso, também se enquadra como um dos elementos fundamentais (PIESSENS, 2021).

Qualquer tipo de erro de um sistema que envolva pelo menos um desses três pilares, pode gerar um falha de segurança. A falha de segurança nada mais é do que um cenário onde o sistema não atinge o seu objetivo de segurança, sendo uma vulnerabilidade a principal responsável por esse acontecimento. Falar sobre vulnerabilidade e segurança só tem sentido concreto quando o objetivo de segurança está bem definido, dessa forma é possível avaliar o que foi uma falha ou não, entretanto não é comum que os projetos tenham isso tão bem definido como os objetivos de usabilidade ou performance, por exemplo (PIESSENS, 2021).

Por conta disso, o objetivo de segurança dos softwares atuais pode se configurar de forma indireta devido à existência de uma classe de *bugs* conhecidos pela comunidade. A partir deles se tem o conhecimento de que ao ser realizado determinado ataque, uma ação não desejada do sistema pode vir a ocorrer. Essas classes também são chamadas de vulnerabilidades de implementação e serão detalhas a seguir (PIESSENS, 2021).

2.2 Vulnerabilidades

Como discutido anteriormente, o termo vulnerabilidade de implementação é utilizado para destacar dois principais cenários: *bugs* que permitam com que um atacante viole um objetivo de segurança, ou classes de *bugs* que possibilitam a utilização de técnicas de ataques. Essas vulnerabilidades tendem a ser ocasionadas pela existência de práticas não seguras na escrita do código ou pela presença de algum tipo de vulnerabilidade na tecnologia ou em dependência externa utilizada dentro do projeto (PIESSENS, 2021).

Existe hoje uma lista pública de vulnerabilidades chamada *Common Vulnerabilities and Exposures* (CVE) que apresenta e descreve vulnerabilidades comuns em componentes de softwares amplamente utilizados no mercado (PIESSENS, 2021). No momento deste estudo, existem mais de 200 mil tipos de vulnerabilidades listadas. Um número tão expressivo como o apresentado acima diz por si só o prejuízo e riscos que um projeto assume ao não considerar as práticas relacionas à segurança de software. A seguir será comentado um pouco sobre alguns dos grupos de vulnerabilidades existentes.

2.2.1 Categoria de Vulnerabilidades

As vulnerabilidades podem ser classificadas de acordo com a sua origem ou com base na parte específica do sistema computacional de onde foi possível explorar algo do sistema (PIESSENS, 2021). A seguir serão apresentadas algumas das principais categorias de vulnerabilidades.

2.2.1.1 Vulnerabilidades no Gerenciamento de Memória

A unidade de processamento de gráficos, também conhecida popularmente como GPUs tornou-se uma peça fundamental no ambiente computacional, sendo responsável por lidar com tarefas e suportar diferentes tipos de arquiteturas, como sistemas paralelos (ZHOU et al., 2016). Além disso, a GPU desempenha um papel fundamental em qualquer tipo de programa hoje desenvolvido. Devido a sua importância, ela é um componente que merece bastante atenção, mas muitas vezes o gerenciamento da sua memória é negligenciado, abrindo portas para que os invasores recuperem dados da memória bruta, deixado por processos anteriores, sem ter nenhum tipo de privilégio (ZHOU et al., 2016).

Com o avanço das linguagens de programação, tornou-se comum o suporte a estados mutáveis, isto é, variáveis que mudam de tipo, ou vetores que aumentam de tamanho, por conta disso existem linguagens que realizam o trabalho de alocar a memória e gerenciar tudo isso por conta própria, como também existem linguagens que passaram essa responsabilidade para o desenvolvedor. Não é por acaso que linguagens como C ou C++ passaram a ser chamadas de linguagens de memória não segura, uma vez que abriram precedentes para diferentes vulnerabilidades.

Quando se fala de vulnerabilidade e memória, existem dois principais tipos: vulnerabilidade de espaço que ocorre quando o programa tenta acessar um espaço além do reservado para determinada variável, ou vulnerabilidade temporal, em que uma célula de memória é acessada depois de já ter sido desalocada (PIESSENS, 2021).

Com essas oportunidades, o atacante consegue explorar esses pontos para acessar locais de memória que são reservadas para outros processos, podendo trazer sérios riscos e danos a um determinado sistema (ZHOU et al., 2016). Um outro exemplo seria acessar áreas de memória reservadas para variáveis de segurança ou chaves de criptografia, por exemplo (PIESSENS, 2021).

2.2.1.2 Vulnerabilidades de Injeção

Nos últimos anos, as aplicações passaram de sistemas que serviam apenas para leitura ou apresentação de informação para sistemas que interagem de maneira complexa e específica com os usuários, o que foi muito bom e marcou boa parte do avanço tecnológico nos últimos anos. Por outro lado, esse avanço permitiu que novos ataques fossem pensados e realizados em meio a esse ambiente tão popular (DEEPA; THILAGAM, 2016).

Junto a isso, além da interação com o usuário, os programas passaram a interagir com outros programas, sendo necessário algumas vezes a construção de uma estrutura de *output*, isto é, uma página HTML, uma *query* SQL para ser consumida por uma base de dados, entre outros casos (PIESSENS, 2021). Com isso, a interação com o usuário misturada com a interação com outros programas permitiu que vulnerabilidade de injeção surgisse.

Dessa forma, o atacante utiliza de inputs ou elementos de interação para enviar códigos ou entradas maliciosas que acabam por interagir com essas estruturas geradas pelo programa e literalmente injetam um *script*, código ou algo do tipo com o objetivo de violar a segurança do sistema e obter outro comportamento que não seja o comportamento "normal" da aplicação.

Vulnerabilidade de injeção é extremamente perigosa, podendo apagar usuários de um sistema, obter informações privilegiadas de uma base, ou até mesmo acessar páginas restritas a partir do código malicioso injetado. Existem alguns tipos de injeção como XPaTH, HTML, SQL, CSS, PostScript (PIESSENS, 2021).

2.2.1.3 Vulnerabilidades de Condição de Corrida

Com o avanço da tecnologia e do poder computacional, cada vez mais tarefas pesadas surgiam para que fossem solucionadas através desses recursos. Buscando sempre uma otimização, sistemas paralelos e distribuídos trouxeram uma ótima alternativa para lidar com problemas de segurança ou problemas de processamento pesado (CARR;

MAYO; SHENE, 2001). Com isso, um dos pontos fundamentais para o funcionamento de tal arquitetura é a utilização de endereços de memórias compartilhados, para que assim os sistemas, ou as *threads* de um programa consigam se comunicar de maneira correta. Semáforos e *locks* passaram a ser usados como alternativa para o controle de acesso e escrita de endereços de memória compartilhados (NETZER; MILLER, 1992).

A vulnerabilidade em cima das condições de corrida também é conhecido como *bugs* de concorrência. Dessa forma, a depender da ordem dos processos ou *threads* que aguardam para utilizar um espaço de memória compartilhado, o resultado obtido pode ser diferente do esperado. Esse tipo de vulnerabilidade pode ocasionar corrupção de arquivos além de uso excessivo do sistema. Por exemplo, um atacante pode votar diversas vezes em uma enquete em que o voto deve ser único devido à concorrência dos processos (PIESSENS, 2021).

2.2.1.4 Vulnerabilidades de API

As APIs são interfaces utilizadas para a realizar a comunicação entre dois programas diferentes a partir de protocolos bem definidos (BHUIYAN et al., 2018). Uma vez que as APIs, normalmente, são serviços oferecidos por terceiros, é necessário garantir que a segurança e o funcionamento do software não sejam afetados por conta disso, uma vez que os desenvolvedores das APIs podem não ter trabalhado algum ponto de segurança que seja fundamental o software que a utiliza.

A partir da violação da interface estabelecida entre o cliente e a API, o cliente entra em um estado de tratamento de erro, que pode ter sido bem implementado ou não, realizando o tratamento correto do erro em questão. Não se preocupar com isso pode abrir oportunidades para que o atacante explore as vulnerabilidades e aberturas que um programa deixa por não lidar com um erro vindo da API (PIESSENS, 2021).

Um outro caso interessante são as mensagens de erro ou os *status code* retornados pela API. A depender da mensagem retornada, um atacante pode verificar que um determinado email tem conta em um sistema e só inseriu a senha de incorreta, isso pode ser um vazamento de dado caso o proprietário do email não queira que as pessoas saibam que ele tem uma conta em determinada plataforma.

2.2.2 Prevenção

É importante entender a causa raiz de um problema para que assim possa ser prevenido. Foi comentado anteriormente que nem sempre é tão simples identificar uma vulnerabilidade ou entender onde o problema começou. Por conta disso, é importante entender a diferença entre os erros que param a execução do programa imediatamente, quebram a aplicação e os erros que passam despercebidos por um determinado tempo. Os

erros que não são percebidos são extremamente perigosos, uma vez que o programa pode ter um comportamento arbitrário após o acontecimento de um erro silencioso (PIESSENS, 2021).

A melhor forma de prevenir uma vulnerabilidade é não inseri-la no código. Uma vulnerabilidade que nunca é acrescentada, nunca precisa ser descoberta (WILLIAMS; MCGRAW; MIGUES, 2018). Como o problema veio crescendo nos últimos anos, as linguagens de programação passaram a oferecer um suporte melhor na prevenção de vulnerabilidades de implementação (PIESSENS, 2021). Está cada vez mais comum ver *warnings* e ferramentas que realizam uma inspeção automatizada para evitar essas vulnerabilidades acrescentadas ao projeto durante a sua implementação.

Ainda assim, é cada vez mais necessário, devido ao crescimento de novas tecnologias e o surgimento diário de novas soluções na área de tecnologia, que o desenvolvedor tenha um conhecimento concreto das oportunidades que um atacante normalmente explora. Como exemplo, a linguagem Haskell não lida com memória mutável ou checagem dinâmica de dados, por outro lado o Python é uma linguagem extremamente fundamentada na checagem dinâmica (PIESSENS, 2021). Por conta disso, não se pode esperar que as linguagens de programação resguardem o software das vulnerabilidades existentes.

Utilizar boas práticas e práticas seguras de desenvolvimento de software tem se tornado a alternativa mais eficiente para prevenir a introdução de vulnerabilidades no código. O grande problema gira em torno dos desenvolvedores não focarem desde o início nas potenciais vulnerabilidades que o seu código pode trazer para o projeto. A construção dessas práticas e modelos vem da discussão da própria comunidade que cria padrões e regras a partir de experiências passadas e problemas que já ocorreram. Também existem padrões de práticas seguras descritas por linguagens de programação como o SEI CERT ou o guia do MISRA (PIESSENS, 2021).

2.2.3 Detecção

É muito comum existirem situações onde não é possível prevenir uma classe de vulnerabilidades de ser introduzida no *software*. Quando não há possibilidade de escolha da linguagem de programação, por exemplo, o programa já está sujeito às classes de vulnerabilidades presentes naquela linguagem. Por conta disso, é necessária a realização de processos e práticas responsáveis por detectar vulnerabilidades em fase de desenvolvimento, testes e manutenção (PIESSENS, 2021).

É importante destacar que existem tipos de vulnerabilidades, assim como também existem vulnerabilidades mais perigosas do que outras. Quando se fala de um sistema em funcionamento, é comum existir diferentes camadas dentro dessa aplicação: cliente, servidor, API, rede, processador, sistema operacional, entre outras. Os bugs encontrados

podem estar em qualquer uma dessas camadas, os que estão cada vez mais perto do sistema operacional acabam sendo os mais perigosos uma vez que o atacante pode executar um código malicioso direto no sistema inteiro e portanto são os que merecem mais atenção (ERDőDI; JOSANG, 2020).

As empresas atualmente vêm dando mais valor para as práticas de detecção de vulnerabilidades antes que venham a se tornar ameaças reais para o sistema, uma vez que os ataques têm sido recorrentes. Para se ter uma ideia do crescimento de ataques, as empresas passaram de um gasto para lidar com crimes cibernéticos de aproximadamente U$ 1.4 milhões em 2017, para U$ 13 milhões em 2018 (HANIF et al., 2021). Um aumento de aproximadamente 928%.

Com o crescimento dos ataques vem aumentando também o interesse dos pesquisadores na área. Existem hoje duas abordagens diferentes para lidar com a detecção de vulnerabilidades: convencional e por *machine learning* e *data-mining*. A abordagem convencional lida com as análises estáticas, dinâmicas e híbridas. A abordagem computacional lida com predições modeladas com base em métricas do software, detecção de anomalias e reconhecimento de códigos vulneráveis (HANIF et al., 2021).

De acordo com Hanif et al. (2021), 74.4% da comunidade de software se interessa pela abordagem de *Machine Learning* para detecção de vulnerabilidades. Dentro desse número, 31.11% preferem a utilização de *Deep Leanrning* utilizando redes neurais como centro da arquitetura e 28.89% a utilização da técnica de aprendizado supervisionado. Por outro lado, os 25.6% restantes que preferem a utilização da abordagem convencional se concentram principalmente na técnica de análise estática, em que o código é examinado sem que seja executado. Uma grande parte também prefere a técnica de análise híbrida. Curiosamente, a técnica de análise dinâmica por si só não é tão popular entre os desenvolvedores. O estudo realizado utilizou sete diferentes técnicas para a abordagem de *Machine Learning* e oito para a abordagem convencional.

Sendo assim, é perceptível que o rumo das detecções e prevenções serão atividades a cargo das máquinas em um futuro não tão distante, mas ainda assim a supervisão humana será sempre necessária para lidar com novos modelos, padrões e categorias de vulnerabilidades até que se estabeleça uma boa confiabilidade com os algoritmos e técnicas computacionais utilizadas.

2.3 Processos de ciclo de vida de software seguro

Os processos de ciclo de vida de software seguro são abordagens que visam garantir a segurança de um produto. Em vez de lidar de forma reativa com os sintomas de softwares inseguros e mal projetados, esses processos buscam tratar a raiz do problema de forma mais eficaz. Existem alguns processos e modelos que abrangem segurança de software

durante todo o ciclo de desenvolvimento do produto (PIESSENS, 2021).

2.3.1 Ciclo de vida de desenvolvimento de software seguro (SSDLC)

O processo chamado *Secure Software Development Life Cycle*, que significa "Ciclo de vida de desenvolvimento de software seguro", é um *framework* que mapeia todo o processo de desenvolvimento, de maneira eficiente e eficaz. O SSDLC incorpora segurança em todas as fases do ciclo de vida do desenvolvimento. Esse processo envolve a integração de práticas de segurança sólidas em conjunto com os aspectos funcionais do desenvolvimento, garantindo a sua proteção (AQUASEC, 2021).

A implementação SSDLC influencia todas as etapas do processo de desenvolvimento de software, exigindo uma abordagem centrada na entrega segura, levantando problemas nas fases de requisitos e desenvolvimento à medida que são descobertos (SNYK, 2020). O SSDLC varia em termos de etapas, podendo alguns modelos possuir cinco etapas principais, enquanto outros seis ou mais, dependendo da abordagem adotada. O modelo descrito na Figura 1 (SNYK, 2020) possui cinco etapas, descritas a seguir:

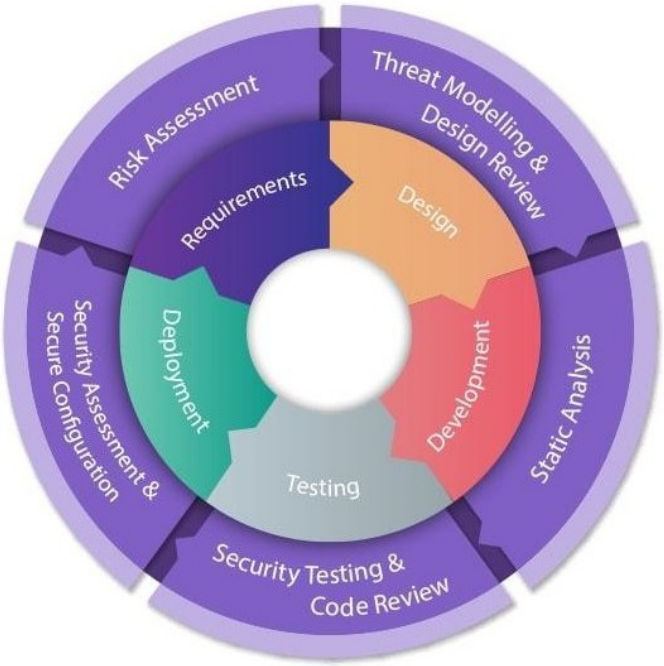

Figura 1 – O Modelo SSDLC (SNYK, 2020)

- ***Requirements*** **(Requisitos)** - Nessa etapa, os requisitos são coletados de várias partes interessadas. É importante identificar quaisquer considerações de segurança para os requisitos funcionais reunidos e as ameaças potenciais. Uma ação investigativa é realizada para encontrar, reconhecer e descrever os riscos de segurança.

- **Design** - Etapa que traduz os requisitos e faz com que eles sejam aplicados. Aqui, os requisitos funcionais geralmente descrevem o que deve acontecer, diferente dos requisitos que descrevem o que não deveria acontecer em relação a sua segurança. Nessa fase, são definidos os padrões para codificação segura e padrões de arquitetura.

- ***Development* (Desenvolvimento)** - Etapa em que o design é implementado, tornando-se realidade. As preocupações mudam comumente de forma a assegurar que o código esteja sendo escrito com segurança. Etapa que possui a análise estática, que é uma revisão automatizada que examina o código em busca de violações de boas práticas de programação, vulnerabilidades de segurança, entre outros.

- ***Testing* (Teste)** - Etapa em que o projeto passa por um ciclo de testes para garantir que ele atende ao design e aos requisitos iniciais. Nessa fase é onde tem a adição de testes automatizados para detectar possíveis falhas de segurança.

- ***Deployment* (Implantação)** - Quando a aplicação é lançada, podem haver vulnerabilidades que passaram despercebidas no desenvolvimento do software. Essas vulnerabilidades precisam ser corrigidas pela equipe de desenvolvimento, essas falhas podem ser descobertas pelos também por meio de testes, como o teste de penetração. Nessa etapa, é bem importante ser o mais automatizada possível, para uma identificação e correção de falhas mais rápido possível.

2.3.2 Construindo segurança no modelo de maturidade (BSIMM)

O *Building Security In Maturity Model* é um modelo que descreve e estabelece uma referência inicial para as atividades observadas em iniciativas de segurança de software. O BSIMM é um estudo que quantifica práticas de diferentes organizações e descreve suas práticas comuns de segurança. O modelo consiste em 12 práticas organizadas em quatro domínios (SYNOPSYS, 2019), como mostra a Figura 2. Os domínios e práticas são descritos a seguir (SYNOPSYS, 2022).

- ***Governance* (Governança)** - Práticas que auxiliam na organização, gerenciamento e medição de uma iniciativa de segurança de software.

  - ***Strategy and Metrics* (Estratégia e métricas)** - Abrange planejamento, atribuição de papéis e responsabilidades, onde é definido a estratégia de segurança e estabelece métricas.

  - ***Compliance and Policy* (Conformidade e Política)** - Está focada na identificação de controles para regimes de conformidade, ajudando a gerenciar e a definir a política de segurança de software. Ou seja, parte de regulação e políticas para o gerenciamento seguro do produto.

Governance	Intelligence	SSDL Touchpoints	Deployment
Strategy and Metrics	Attack Models	Architecture Analysis	Penetration Testing
Compliance and Policy	Security Features and Design	Code Review	Software Environment
Training	Standards and Requirements	Security Testing	Configuration Management and Vulnerability Management

Figura 2 – O modelo BSIMM (MCGRAW, 2009)

  - ***Training*** **(treinamento)** - Etapa da educação e conscientização sobre segurança de software fornecida aos funcionários e membros da equipe envolvidos no desenvolvimento do produto. O treinamento sempre desempenhou um papel crítico na segurança de software porque as partes interessadas como engenharia, operações e outros grupos, geralmente começam com pouco conhecimento de segurança.

- ***Intelligence (Inteligência)*** - Práticas que resultam na criação de conhecimento corporativo utilizado na execução de atividades de segurança de software em toda a organização.

  - ***Attack Models*** **(Modelos de ataque)** - Os modelos de ataque capturam informações usadas para pensar como um invasor. São usados para identificar possíveis pontos fracos e vulnerabilidades do software.
  - ***Security Features and Design*** **(Recursos de segurança e design)** - É responsável por criar padrões de segurança utilizáveis para os principais controles de segurança (atendendo aos padrões definidos na prática de padrões e requisitos).
  - ***Standards and Requirements*** **(Padrões e Requisitos)** - Envolve obter requisitos explícitos de segurança de software da organização, criar padrões para os principais controles de segurança e criando um processo de revisão de padrões.

- ***SSDL Touchpoints*** **(Pontos de contato SSDL)** - Práticas relacionadas à análise e garantia de artefatos e processos específicos no desenvolvimento de software.

  - ***Architecture Analysis*** **(Análise de arquitetura)** - Abrange a captura da arquitetura de software em diagramas concisos, aplicando listas de riscos e

ameaças. Ou seja, faz a análise da arquitetura de software em busca de riscos e vulnerabilidades.

- ***Code Review*** **(Revisão de código)** - Faz a análise do código-fonte em busca de falhas de segurança, onde inclui o uso de ferramentas de revisão de código (por exemplo, análise estática).
- ***Security Testing*** **(Teste de segurança)** - Essa etapa está preocupada com a descoberta de defeitos no pré-lançamento do produto, incluindo a integração de segurança em processos padrão de controle de qualidade. São realizados testes de segurança abrangentes para identificar falhas e vulnerabilidades no software.

- ***Deployment*** **(Implantação)** - Práticas que interagem com organizações de segurança de rede e manutenção de software.
  - ***Penetration Testing*** **(Teste de penetração)** - O teste de penetração concentra-se em vulnerabilidades no código de pré-produção e produção, fornecendo informações em tempo real para gerenciamento e mitigação de defeitos. Processo no qual especialistas em segurança simulam ataques reais contra o sistema para identificar vulnerabilidades e pontos fracos.
  - ***Software Environment*** **(Ambiente de software)** - Lida com *patches* (programa de computador criado para atualizar ou corrigir um software de forma a melhorar sua usabilidade ou performance) de sistema operacional e plataforma, documentação, entre outros. Essa etapa envolve o estabelecimento e a manutenção de um ambiente de software seguro, no qual o software é implantado e executado.
  - ***Configuration Management and Vulnerability Management*** **(Gerenciamento de configuração e gerenciamento de vulnerabilidade)** - Preocupa-se com processos operacionais, controle de versão, rastreamento e correção de defeitos e tratamento de incidentes. Essa etapa envolve a implementação de práticas e processos eficazes para o gerenciamento de configuração adequada e o gerenciamento de vulnerabilidades ao implantar o software.

2.3.3 Modelo de Maturidade de Garantia de Software (SAMM)

O termo *Software Assurance Maturity Model* que significa "Modelo de Maturidade de Garantia de Software", é uma prática desenvolvida pela fundação OWASP[1] para auxiliar as organizações a melhorar a segurança de seus processos de desenvolvimento de software (OWASP, 2020). Essa metodologia fornece uma estrutura com passo a passo de

[1] https://owasp.org/

como organizações devem seguir para implementá-lo. O primeiro modelo foi feito em 2009 agrupados em cinco funções: Governança, Construção, Verificação e Implantação.

Após cerca de 10 anos, foi feito alterações para a versão dois. O modelo é baseado em 15 práticas de segurança agrupados em cinco funções. Cada prática contém um grupo de atividades estruturadas em três níveis de maturidade e cada um possui dois fluxos. Os fluxos cobrem diferentes aspectos de uma prática e cada um deles têm um objetivo diferente alinhados com os níveis de maturidade. A Figura 3 ilustra o modelo.

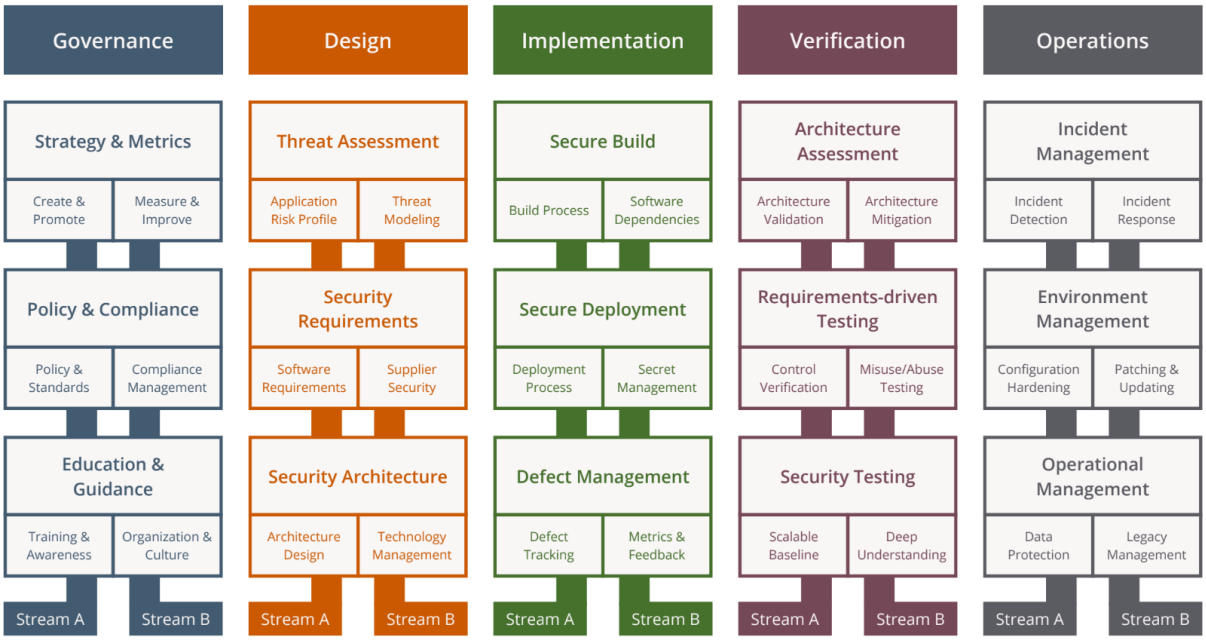

Figura 3 – O modelo SAMM v2 (OWASP, 2020)

Essa versão foi projetada para facilitar atualizações frequentes, sendo implementadas por meio de pequenas alterações incrementais em partes específicas do modelo. Os três níveis de maturidade permanecem como estavam: o nível 1 é a implementação inicial, o nível 2 a realização estruturada e o nível 3, operação otimizada (MEUCCI, 2020). As cinco principais funções de negócios são explicadas abaixo, com base no documento do site oficial da OWASP (OWASP, 2020):

- ***Governance* (Governança)** - A Governança abrange os processos e atividades relacionados à forma como uma organização gerencia as atividades de desenvolvimento de software como um todo. Ela envolve a definição de estratégias de segurança, se tornando uma parte fundamental, pois nela que estabelece uma estrutura eficaz que promove a segurança em todo o ciclo de vida do desenvolvimento de software. Como é uma fase de definições, seus 3 níveis de maturidade são: Estratégia e métricas, Política e Conformidade, Educação e Orientação.

- **Design** - Em geral, esse processo inclui coleta de requisitos de segurança, aplicação de princípios de design seguro, a identificação e mitigação de ameaças e vulnerabilidade, especificação de arquitetura de alto nível e design detalhado. O objetivo dessa etapa é construir um software seguro para evitar problemas de segurança no produto final. Seus três níveis de maturidade são: Avaliação de ameaça, Requisitos de segurança e Arquitetura de segurança.

- ***Implementation*** **(Implementação)** - A implementação é focada nos processos e atividades relacionados a como uma organização constrói e implanta componentes de software e seus defeitos relacionados. Nesta fase está a codificação da aplicação, com ênfase em desenvolver o software de forma segura. Os três níveis de maturidade são: Construção segura, Desenvolvimento seguro e Gestão de defeitos.

- ***Verification*** **(Verificação)** - A verificação se concentra nos processos de como uma organização verifica e testa o que foi produzido no desenvolvimento do software. Nessa etapa, são realizados os testes de segurança, análise do código, revisões de arquitetura e outras atividades que garante a proteção do produto. Os três níveis de maturidade dessa etapa são: Avaliação de arquitetura, Testes orientados a requisitos e Testes de segurança.

- ***Operations*** **(Operações)** - Essa parte abrange as atividades necessárias para garantir que a confidencialidade, a integridade e a disponibilidade sejam mantidas durante todo o tempo de vida operacional do projeto. Nessa fase, são adotadas medidas para proteger o software em tempo de execução, abrangendo o monitoramento da integridade do sistema. Níveis de maturidade dessa etapa são: Gestão de incidentes, Gerenciamento de ambiente e Gerenciamento operacional.

2.3.4 Ciclo de vida de desenvolvimento de segurança (SDL)

O *Security Development Lifecycle* (SDL) é um processo que tem como objetivo incorporar a segurança em cada fase do ciclo de vida do desenvolvimento de software. Esse processo foi desenvolvido pela Microsoft, tendo como um dos seus objetivos identificar e eliminar vulnerabilidades de segurança antes que o software seja implantado. O SDL faz modificação dos processos de uma organização de desenvolvimento, integrando medidas que levam à melhoria de software (LIPNER; HOWARD, 2010).

Para compreender melhor o SDL, é importante entender como ele se encaixa no processo de desenvolvimento de software, considerando diversas fases como: treinamento, requisitos, design, implementação, verificação, versão e resposta. Esse modelo se baseia em três principais componentes: educação, melhoria de processos e responsabilidade. Para um melhor entendimento desse processo, a Figura 4 apresenta uma visão geral do método.

Figura 4 – O Microsoft SDL - Simplificado (CZECHOWSKI et al., 2023)

A seguir cada etapa do SDL é descrita com base nas informações encontradas no site oficial da Microsoft[2] (MICROSOFT, 2011):

- **Pré-requisitos do SDL - Treinamento de segurança**:

  Prática do SDL 1: Requisitos de treinamento

  Todos os membros da equipe de desenvolvimento devem receber treinamento sobre fundamentos básicos de segurança e de privacidade. Pessoas com funções técnicas como gerente, desenvolvedor e testadores devem receber o treinamento ao menos uma vez por ano.

  O treinamento deve conter conceitos fundamentais, incluindo tópicos como:

  - **Design de segurança**: Defesa em propriedade, padrões seguros;
  - **Modelagem de ameaças**: Visão geral da modelagem de ameaças, implicações de design de um modelo de ameaças;
  - **Codificação de segurança**: Estouros de *buffer*, erros aritméticos inteiros;
  - **Teste de segurança**: Avaliação de riscos, métodos de teste de segurança;
  - **Privacidade**: Avaliação de riscos, tipos de dados importantes de privacidade.

- **Fase um - Requisitos**:

  Prática do SDL 2: Requisitos de segurança

  A análise de requisitos e de privacidade é realizada no primeiro esboço do projeto, na parte de planejamento. Conforme a aplicação é criada há um monitoramento de vulnerabilidades de segurança para garantir que a aplicação esteja sempre protegida contra eventuais ameaças.

  Prática do SDL 3: Portões de qualidade/Barras de erros

  Os portões de qualidade/barras de erros são utilizados para limitar níveis mínimos aceitáveis de qualidade de software. Essa definição no início do projeto auxilia a

[2] https://learn.microsoft.com/pt-br/docs/

entender os riscos e problemas de segurança da plataforma durante o seu desenvolvimento. Uma vez esse limite sendo estabelecido, não deve ser cedido.

Prática do SDL 4: Avaliação de riscos de segurança e de privacidade

São processos obrigatórios que identificam os aspectos funcionais do software que requerem uma análise profunda. Essas avaliações devem incluir perguntas como:

- Quais partes do projeto requerem modelos de ameaças antes da liberação?
- Quais partes do projeto requerem revisões do design de segurança antes da liberação?
- Quais partes do projeto (se houver) exigirão um teste de penetração por um grupo de comum acordo que seja externo à equipe do projeto?
- O que é a Classificação de impacto de privacidade? (Classificados por: **P1** Risco de privacidade alto, **P2** Risco de privacidade moderado, e **P3** Risco de privacidade baixo)

- **Fase dois - Design**:

  Prática do SDL 5: Requisitos de design

  É muito importante considerar questões de segurança e privacidade durante a fase de design, pois essa mitigação fica muito mais barata durante os estágios iniciais do ciclo de vida de um projeto.

  As especificações de design devem abranger uma descrição detalhada dos diferentes tipos de ataques que os usuários podem estar sujeitos. Além disso, as especificações dessa etapa, devem descrever como devem ser implementas as funcionalidades de forma segura.

  Prática do SDL 6: Redução de superfície de ataque

  A redução de superfície de ataque abrange o fechamento de permissões ou restrições do sistema, concedendo privilégios a determinados usuários e implementando defesas em camadas sempre que possível. Com isto, acaba reduzindo os riscos, pois está sendo dado menos oportunidades aos invasores de entrarem no sistema.

  Prática do SDL 7: Modelagem de ameaças

  Trata-se de uma prática que visa permitir que as equipes de desenvolvimento considerem, documentem e debatam de maneira estruturada as implicações de segurança dos projetos em relação ao ambiente operacional planejado. Essa modelagem é mais um exercício de equipe e representa a tarefa de análise de segurança primária.

- **Fase três - Implementação**:

  Prática do SDL 8: Utilizar ferramentas aprovadas

Todas as equipes de desenvolvimento devem definir uma lista de ferramentas que serão utilizadas no projeto, e essas ferramentas devem ser aprovadas pelo consultor de segurança da equipe. Sendo essas ferramentas utilizadas nas versões mais atuais para uma maior proteção.

Prática do SDL 9: Desaprovar funções não seguras

A equipe deve avaliar todas as funções e APIs que estão determinadas como não seguras e proibi-las. Com esta lista de funções e APIs, devem ser utilizado ferramentas de verificação de código (seguras) para encontrar funções não seguras e substituí-las.

Prática do SDL 10: Análise estática

Essa é uma análise feita examinando o código fonte, em que o objetivo é identificar possíveis erros ou vulnerabilidades. Ou seja, a equipe deve fazer essa análise, porém ela sozinha geralmente é insuficiente, o time deve estar ciente sobre isso e deve estar preparado para acrescentar ferramentas de análise estática ou revisão humana.

- **Fase quatro - Verificação**:

  Prática do SDL 11: Análise de programa dinâmica

  A verificação de tempo de execução de um programa de software é essencial, essa tarefa de verificação requer a especificação de ferramentas que possam monitorar o comportamento software quanto a eventos críticos de segurança. O processo SDL utiliza ferramentas como AppVerifier para atingir níveis de cobertura de segurança desejados.

  Prática do SDL 12: Teste de *fuzzing*

  O teste de *fuzzing* é uma forma especializada de análise dinâmica, onde é utilizado para induzir dados defeituosos ao programa com o objetivo de identificar a forma que o software pode reagir a essas entradas, com isto, fica mais fácil de corrigir essas falhas.

  Prática do SDL 13: Modelo de ameaças e revisão da superfície de ataque

  É comum a aplicação desviar algumas especificações, desse modo, é fundamental reavaliar os modelos de ameaça e a medição da superfície de ataque do mesmo. Com isto, é garantido que as alterações de implementação ou design sejam considerados.

- **Fase cinco - Liberação**:

  Prática do SDL 14: Plano de resposta de incidentes

  Deve ser elaborado um plano de resposta de incidentes, pois mesmo os programas sem vulnerabilidades estão sujeitos a ameaças que surgem com o tempo. Esse plano deve incluir:

- Uma equipe de Engenharia sustentada ou caso a equipe seja muito pequena um Plano de resposta de emergência;
- Contatos de autoridades para denúncia;
- Serviços de segurança planejados para códigos legados provenientes de diferentes grupos internos da organização;
- Planos de serviço de segurança para código de terceiros licenciados.

Prática do SDL 15: Revisão final de segurança

Uma análise típica da revisão final de segurança, envolve examinar os modelos de ameaça, solicitações de exceção, saída de ferramentas e o desempenho em relação aos critérios de qualidade ou limites de erros estabelecidos anteriormente. Essa revisão é realizada pelo consultor de segurança juntamente com a equipe de desenvolvedor e os líderes das equipes de segurança.

A revisão resulta em três tipos de resultados:

- **Revisão aprovada**: Todos os problemas de segurança foram resolvidos.
- **Revisão aprovada com exceções**: Todos os problemas de segurança foram resolvidos, mas tem problemas que não podem ser abordados, porém devem ser registrados e resolvidos posteriormente.
- **Revisão com escalação**: Quando a equipe não satisfez todos os requisitos, o consultor acaba não aceitando o projeto.

Prática do SDL 16: Liberar/Arquivar

A liberação do projeto depende da conclusão do SDL. Deve ser certificado que a equipe cumpriu todos os requisitos necessários para a liberação do produto. Além disso, todas as informações e dados pertinentes devem ser arquivados para permitir a instalação do software após seu lançamento, isso inclui todas as especificações como documentos, código fonte, entre outros.

2.4 Stack Overflow e Fóruns de Perguntas e Respostas

O Stack Overflow (SO), IT Security e Software Engineering, serão a base da extração dos dados e das análises feitas neste estudo. Sendo assim, é importante entender como essas ferramentas funcionam para um melhor entendimento do que pode ser feito.

Desenvolvedores gastam a maior parte do seu tempo em manutenções do software, que tem um custo aproximado de 85% a 90% no custo total do desenvolvimento. Tendo isso em vista, os serviços de pergunta e resposta, como o SO, atacam justamente a documentação e o compartilhamento de conhecimento para que os problemas de manutenção

sejam tarefas mais fáceis no dia a dia do desenvolvedor (PONZANELLI; BACCHELLI; LANZA, 2013).

Os fóruns de pesquisa oferecem uma plataforma de discussão de assuntos técnicos referente a um problema em específico. Geralmente um usuário realiza uma pergunta sobre algo que não tenha ficado tão claro pela documentação, ou uma tarefa que é difícil de realizar, ou até mesmo sobre algo que ele tenha curiosidade, e então pessoas de outros projetos e que possivelmente tenham mais conhecimento nesse tópico levantado acabam respondendo a dúvida. Mesmo alguns estudos dizendo que as respostas obtidas nesses serviços não possuem um nível técnico alto, a comunidade viu esses mesmos serviços suprirem milhões de dúvidas de tal maneira que passaram a desempenhar um papel de documentação e suporte aos desenvolvedores (PONZANELLI; BACCHELLI; LANZA, 2013).

Dois pontos são extremamente fundamentais para o estudo em questão. Nem sempre uma pergunta possui uma resposta aceita dentro da plataforma, ou por falta de tempo, ou por falta de conhecimento dos usuários que visualizaram a questão. O site funciona como uma discussão, então qualquer usuário pode colocar uma resposta a determinada dúvida levantada, mas apenas uma resposta é dada como aceita, sendo a resposta correta para determinada pergunta. O aceite da questão é feito pelo usuário que perguntou, assim ele determina qual resposta realmente ajudou na resolução do problema.

O segundo ponto que merece destaque é a utilização de *tags* na realização de uma pergunta. As *tags* são uma forma de categorizar uma determinada pergunta de acordo com o seu tema. Em setembro de 2014 existiam mais de 38.000 dentro do SO, por conta disso foi feito um trabalho para correlacionar as *tags* sinônimas para que assim facilitasse e reduzisse a quantidade de diferentes *tags* existentes na plataforma (BEYER; PINZGER, 2015) . Essa funcionalidade é muito importante para unificar perguntas referentes ao mesmo assunto, dessa forma o estudo em questão pode utilizá-las para averiguar quais são as mais utilizadas dentro de um determinado assunto ou categoria de Segurança de Software.

2.5 Alocação de Dirichlet Latente (LDA)

O LDA é um método de aprendizado de máquina, não supervisionado, utilizado para identificar e extrair tópicos em coleções de documentos. Nesse modelo, os termos de cada documento são as variáveis observáveis, enquanto as distribuições de tópicos são as variáveis não observáveis. A distribuição de Dirichlet (distribuição de probabilidade contínua multivariada) é utilizada para amostrar as distribuições de tópicos. No processo gerativo, as amostras da distribuição de Dirichlet são usadas para atribuir palavras de diferentes tópicos aos documentos. Com isto, o LDA tem a intenção de alocar tópicos

latentes de acordo com a distribuição de Dirichlet (FALEIROS, 2016).

O processo generativo pode ser representado por uma rede Bayesiana (modelo estatístico utilizado para representar relações probabilísticas entre um conjunto de variáveis), como mostra a Figura 5.

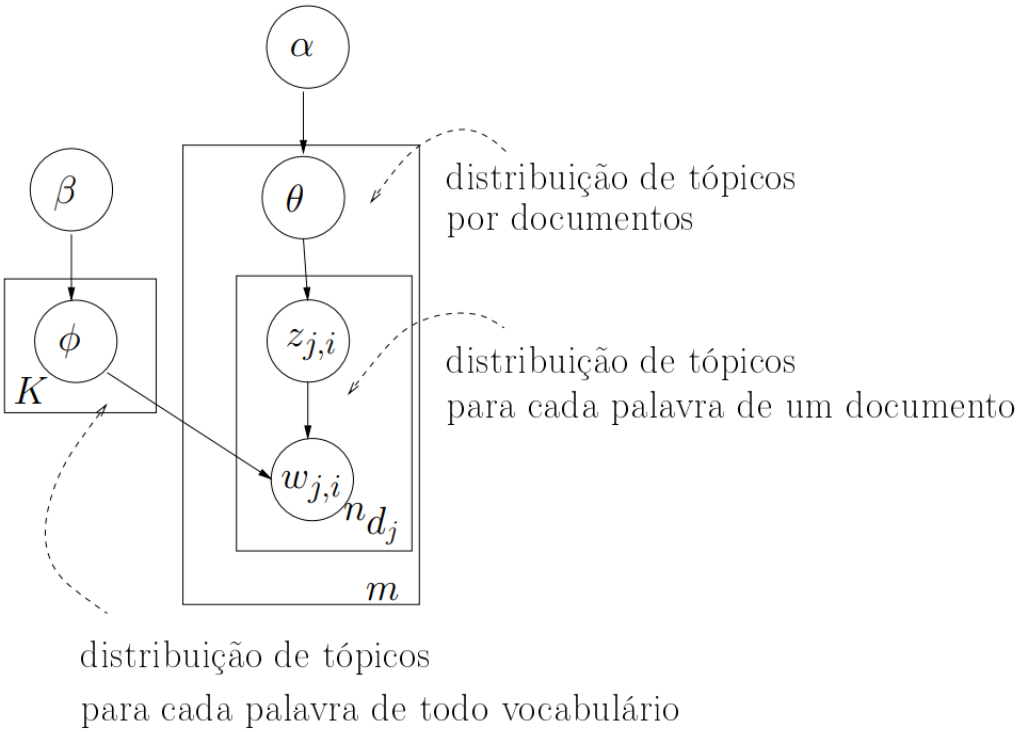

Figura 5 – Modelo Gráfico do LDA (FALEIROS, 2016)

Conforme a notação de (FALEIROS, 2016):

- K - número de tópicos.
- n - número de palavras do vocabulário.
- m - número de documentos.
- n_{d_j} - número de palavras em um documento d_j, onde $1 \leq j \leq m$.
- θ - distribuição de tópicos por documentos.
- ϕ - distribuição dos tópicos sobre as palavras do vocabulário.
- θ_j - vetor com a proporção dos tópicos para o documento d_j, onde $1 \leq j \leq m$.
- ϕ_k - vetor com a proporção das palavras do vocabulário para o tópico k, onde $1 \leq k \leq K$.
- α - priore da distribuição de Dirichlet, relacionada à distribuição documento-termo.

- β - priore da distribuição de Dirichlet, relacionada à distribuição tópico-palavra.
- w_i - i-ésima palavra do vocabulário, onde $1 \leq i \leq n$.
- $w_{j,i}$ - palavra w_i observada no documento d_j, onde $1 \leq j \leq m$ e $1 \leq i \leq n$.
- $z_{j,i}$ - distribuição de tópicos associada à palavra $w_{j,i}$ no documento d_j, onde $1 \leq j \leq m$ e $1 \leq i \leq n$.

Segundo o Faleiros (2016), o modelo Bayesiano é um modelo com três níveis, como mostra a Figura 5. O primeiro nível representa a distribuição de tópicos em toda a coleção de documentos. O segundo nível, distribuição de tópicos para cada documento. O terceiro nível repete a distribuição dos tópicos internamente para as palavras em um documento, sendo possível representar um documento com uma mistura de tópicos.

3 Metodologia

Este capítulo tem como objetivo detalhar a metodologia utilizada para o desenvolvimento da pesquisa. Para isso, a metodologia foi classificada levando em consideração sua natureza, abordagem e procedimentos técnicos.

Por se tratar de um estudo que tem como objetivo a extração e processamento de dados referentes ao assunto, a pesquisa em questão tem uma abordagem quantitativa, a fim de demonstrar em números os resultados obtidos. O objetivo final deste estudo é compreender os principais pontos de lacuna no estudo de Segurança de Software por parte dos desenvolvedores e entusiastas da área. A natureza do estudo é do tipo aplicada, voltada para a compreensão de um problema específico na área de Software. Para alcançar esse objetivo, serão utilizados os seguintes procedimentos técnicos:

- **Pesquisa Bibliográfica**: Assim como um levantamento bibliográfico, desempenha um papel fundamental neste estudo, pois contribui para embasar o conteúdo e material apresentado, permeando todo o processo de pesquisa. É fundamental, portanto, que em qualquer trabalho científico, seja realizada uma pesquisa bibliográfica exaustiva sobre o tema em questão, antes de iniciar a coleta de dados (AMARAL, 2007).

- **Pesquisa Documental**: Procedimento que se utiliza de métodos e técnicas para a apreensão, compreensão e análise de documentos dos mais variados tipos (Sá-SILVA; ALMEIDA; GUINDANI, 2009). É um método de investigação que utiliza documentos como fonte principal de dados. Nesse contexto, serão coletados conteúdos de fóruns e outras informações de materiais que não receberam um tratamento analítico.

3.1 Plano metodológico

As etapas da metodologia estão organizadas em:

- Planejamento da Pesquisa;

- Referências Bibliográficas;

- Plano de ação.

3.2 Planejamento da Pesquisa

Por se tratar de um trabalho de conclusão de curso, seu planejamento foi atrelado às principais entregas durante o processo de desenvolvimento do estudo. A sequência de etapas planejadas para essa pesquisa pode ser vista na Figura 6.

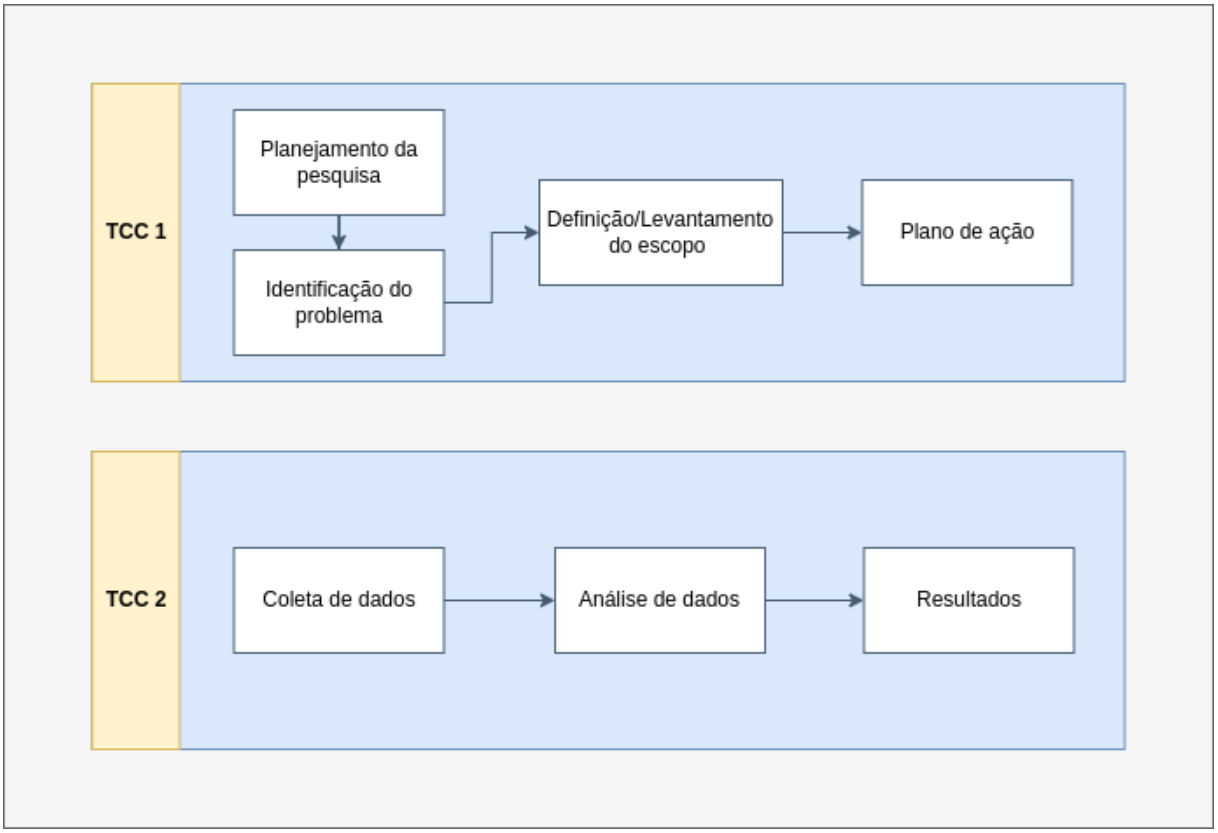

Figura 6 – Etapas do projeto, Fonte: Autores

A primeira etapa do projeto é focada em definir os pontos fundamentais da pesquisa como o planejamento da pesquisa (objetivo e metodologia), identificação do problema, definição do escopo juntamente com a coleta de informações e o plano de ação. Além disso, a primeira etapa contém uma breve introdução da aplicação para que alguns resultados prévios sejam entregues.

A segunda etapa desta pesquisa abrange a coleta/extração e análise de dados, bem como a divulgação dos resultados obtidos, seguindo o que foi definido pelo objetivo na etapa anterior.

3.3 Referências Bibliográficas

Para estabelecer uma base teórica e fundamentada sobre o que foi apresentado nesse estudo, foi realizado uma revisão bibliográfica para suprir essas necessidades. Além

disso foram pesquisados artigos e estudos que seguem uma metodologia similar, para que assim, existisse a possibilidade de comparar resultados e propostas desenvolvidos nesses trabalhos.

Com base na pesquisa conduzida por Alfayez et al. (2023), surgiu a proposta de realização de um estudo similar, mas com foco na área de segurança de software. Três artigos foram fundamentais como base para a realização desta pesquisa, sendo eles:

- Alfayez et al. (2023), onde é realizado um estudo sobre os principais assuntos que os desenvolvedores discutem sobre dívida técnica, assim como das *tags* utilizadas pelos usuários na hora de classificar as suas perguntas. Esse artigo é a base de inspiração desse estudo e referencia boa parte da metodologia utilizada nesse trabalho.
- Rahman (2016), um trabalho similar ao estudo anterior, que trata das questões do SO na área de Segurança de Software, realizado no ano de 2016. Além de fornecer métricas e ferramentas utilizadas para atingir resultados parecidos ao desse trabalho, o artigo em questão serve também para que seja feita uma comparação entre os resultados obtidos em 2016 e os resultados obtidos no presente estudo.
- Yang et al. (2016), este artigo apresenta um estudo em larga escala sobre questões relacionadas à segurança no SO. O objetivo do estudo é fornecer um guia para pesquisadores, educadores e profissionais de segurança, investigando tópicos e tendências relacionados à segurança. Esse estudo ajuda a completar algumas etapas da metodologia do pesquisa atual.

Os três artigos acima são os trabalhos que serviram de base para construção e inspiração do trabalho desenvolvido, apesar disso outros artigos não destacados acima serviram de base para a fundamentação teórica, além de contribuições para metodologia, ideias e organização do estudo. O que este estudo traz de diferente dos estudos apresentados, é que foi utilizado mais gráficos visuais, visando mostrar a evolução das perguntas conforme o tempo, mapeando onde a comunidade continua tendo mais dúvidas. Mostra uma junção dos artigos que estão sendo tomados como base, disponibilizando o código para que outros usuários possam reproduzir os resultados.

Além do referencial bibliográfico, esta etapa também foi responsável por realizar pesquisas e estudos sobre algumas alternativas e variáveis que poderiam mudar o estudo em questão. Sendo assim, os pontos principais que foram definidos antes da realização do estudo foram:

- Definição dos fóruns que seriam utilizados na pesquisa. O SO é hoje o fórum mais conhecido entre os desenvolvedores e se torna essencial para esta pesquisa, uma vez que possui a maior quantidade de dados para análise. O *IT Security* é um fórum de

pesquisa focado na área de segurança, e por isso acreditamos ser interessante ter um fórum mais especializado na pesquisa. Por fim, utilizamos o *Software Engineering* que é focado em metodologias de software e pelo fato de ter sido utilizado pelo estudo de Alfayez et al. (2023).

- Definição do idioma a ser utilizado para a análise das perguntas nos fóruns. Considerando que os fóruns são globais, a maioria das perguntas se encontram na língua inglesa. Dado que haverá um processamento dessas perguntas, o uso de múltiplos idiomas pode dificultar uma análise consistente. Por conta disso foi definido que seria utilizado apenas as perguntas em inglês.

- Definição do escopo de pesquisa de segurança de software. Dentro da área de desenvolvimento existem questões de segurança direcionadas a aplicações web, mobile, *desktop*, entre outras. Foi definido que seria utilizado um contexto geral, uma vez que alguns estão diretamente interligados e essa abordagem não prejudica o estudo.

3.4 Plano de ação

A partir das informações coletadas e dos artigos modelos, foi elaborado um plano de ação para a condução da pesquisa. Esse plano foi dividido da seguinte forma:

1. Extração das perguntas e respostas;
2. Categorização das perguntas com LDA;
3. Análise das *Tags* relacionadas com as perguntas;
4. Identificação das perguntas com categorias mais difíceis;

3.4.1 Extração das perguntas e respostas

O primeiro passo fundamental dentro do plano de ação é a realização de um processo para a extração das perguntas e seus dados dos fóruns. O estudo em questão fez uso do site *StackExchange Data Explorer*[1] que funciona como uma base dados de diferentes fóruns de pesquisa, permitindo que sejam realizadas consultas SQL para extrair e filtrar as questões da maneira que desejar.

No atual estado em que o estudo foi feito, os fóruns que serão utilizados possuem a seguinte quantidade de questões armazenadas: SO possui 58.808.970 questões, SE um total de 240.744 questões e IT Security 186.377 questões.

[1] https://data.stackexchange.com/

O termo *Software Security* foi inicialmente utilizado para realizar uma filtragem preliminar e obter uma visão abrangente das perguntas e respostas relevantes que seriam abordadas posteriormente. Para isso, foram pesquisados todos os *Posts* que mencionavam o termo no corpo ou no título. Com o resultado obtido, foi constatado que seria necessário um refinamento da consulta *SQL* para que o estudo trabalhasse em cima de questões mais apropriadas e consistentes levando em consideração os objetivos do estudo.

Dessa forma, os termos adicionados juntamente com o *Software Security* são: *Vulnerab%* (onde esse mostrará todas as palavras que começam com essa valor), *Web Security*, *Access Control*, *Mobile Security*, *System Security*, e por fim, *tags* que contenham a palavra *security*.

3.4.2 Categorização das perguntas com LDA

Após o levantamento das perguntas e respostas dos sites, foi feita uma categorização das perguntas com o LDA (*Latent Dirichlet Allocation*), proposto por Blei, Ng e Jordan (2003). Esse modelo foi uma ferramenta essencial para realizar a categorização das perguntas de forma automatizada, pois a quantidade de perguntas extraídas é grande, tornando a classificação manual uma tarefa difícil de ser realizada.

Neste trabalho, optou-se por implementar o LDA utilizando a linguagem Python, com a biblioteca gensim [2], devido ao seu suporte simples e direto. Esse modelo envolve etapas como:

- Preparação dos dados: seleção das questões dos fóruns, que foram extraídas anteriormente, levando em consideração aquelas que são relevantes para o estudo em questão;

- Ajuste do Modelo (ou Estimação de Parâmetros): Nessa fase, treina-se o modelo LDA com os dados disponíveis, e é aqui que as distribuições de tópicos por documento e as distribuições de palavras por tópico são calculadas a partir do corpus de documentos. Este ponto é central no processo do LDA, pois é quando o modelo procura revelar a estrutura subjacente dos tópicos nos documentos;

- Inferência de tópicos: Após o treinamento, pode ser utilizado o modelo para inferir os tópicos em novos documentos;

- Avaliação e ajuste: Avaliação dos resultados do modelo, e refinamento caso necessário.

[2] https://pypi.org/project/gensim/

3.4.3 Análise das *Tags* relacionadas com as perguntas

Nos fóruns existem *tags* relacionadas com cada pergunta que o usuário colocou para ser possível identificar mais facilmente o seu tema. Uma pergunta deve conter pelo menos uma *tag* e pode ter no máximo cinco. Para realizar o levantamento das *tags*, foram seguidos dois passos, conforme descrito no artigo Alfayez et al. (2023):

1. Desenvolvimento de um código para identificar quais *tags* são mais utilizadas, e com isto, calcular o número de vezes que as *tags* mais utilizadas são referenciadas nos fóruns. Isso fornece um contexto melhor sobre como o tema de segurança de software é utilizado em todos os fóruns utilizados. Esses dados são mostrados em um gráfico para melhor visualização.

2. Desenvolvimento de um código para identificar o número de ocorrências de cada *tag*, e com isto, calcular o número total de ocorrências de cada *tag* dentro de cada categoria pré-definida. Com isso é possível obter dados sobre quais *tags* são mais relevantes e frequentes em cada categoria. Esses dados são mostrados em um gráfico para melhor visualização.

3.4.4 Identificação das perguntas com categorias mais difíceis

Um tópico sendo popular e difícil de responder, deve receber mais atenção. Identificar esses tópicos pode ajudar os desenvolvedores a ter uma visão mais clara sobre as perguntas mais desafiadoras, para que possam dedicar mais tempo à resolução delas (YANG et al., 2016).

Foram escolhidas duas heurísticas para estimar a dificuldade de cada categoria:

1. A porcentagem de perguntas relacionadas com a categoria de segurança de software que não obtiveram resposta. Essa heurística foi escolhida porque a ausência de uma resposta aceita pode sugerir que o autor da pergunta não encontrou nenhuma resposta adequada e que ele ainda pode estar enfrentando dificuldades (ALFAYEZ et al., 2023).

2. Intervalo de tempo médio para as perguntas obterem respostas aceitas. Para determinar a duração de uma pergunta, será considerado a data de criação da pergunta como o ponto de partida e a data de criação da resposta aceita correspondente como o ponto final. Para obter esse valor, primeiro obtemos o número de respostas e o número de visualizações dos atributos denominados *"AnswerCount"* (contagem de respostas) e *"ViewCount"* (contagem de visualizações), respectivamente, de cada pergunta em um determinado tópico.(YANG et al., 2016)

Para efeito de cálculo, a data de criação da pergunta e a data da resposta aceita foram convertidas para um objeto ***datetime***, que é a representação de uma data e de um horário específico, para que fosse possível extrair um ***timestamp*** referente à quantidade de segundos desta data até o dia 1º de janeiro de 1970. Depois disso, esses tempos foram convertidos para minutos, para que assim seja feito o cálculo final do tempo médio de resposta para cada uma das categorias. O cálculo do tempo médio é mostrado na equação abaixo:

$$MTR = \frac{\sum_{n=1}^{QuestionsN} \frac{RespostaSeg - PerguntaSeg}{60}}{QuestionsN},$$

onde 'MTR' significa *"Média de Tempo de Resposta"*, *"QuestionsN"* é a quantidade total de perguntas dentro de cada categoria, *"RespostaSeg"* equivale a quantidade de segundos desde o dia 1º de janeiro de 1970 e *"PerguntaSeg"* a quantidade de segundos da data de criação da pergunta. Por existir uma grande quantidade de perguntas, esse processo de calcular a média das respostas e das visualizações foi realizado de forma automatizada, por meio do desenvolvimento de um código para tal função.

4 Resultados

Este capítulo apresenta a aplicação do plano de ação da pesquisa os resultados obtidos.

4.1 Extração das perguntas e respostas

Inicialmente tinha sido definido que o termo a ser utilizado para o estudo em questão seria: “Software Security”. Porém, após algumas análises dos resultados obtidos, chegou-se a conclusão que o termo não ofereceu uma quantidade satisfatória de questões para serem analisadas, retornando apenas 171 questões, muito abaixo do esperado.

Em seguida, decidimos prosseguir com a análise do estudo de Rahman (2016), utilizando os conceitos identificados. No entanto, optamos por criar um painel de controle (Dashboard) que apresenta o histórico de todos os anos encontrados nos fóruns. Para isso, ampliamos o período de análise, abrangendo o intervalo de Agosto de 2008 a Outubro de 2023.

Com o propósito de alcançar essa meta, aprimoramos nosso código de extração de perguntas, agora aplicando uma consulta (query) direcionada à busca da *tag* ’%security%’. Isso nos permitiu identificar *tags* que incluem a palavra ’security’ ou suas variantes.

Para seguir com a nossa análise, utilizamos o conjunto de dados dos fóruns selecionados, que está disponível no *Stack Exchange Data*[1]. O código utilizado para essa extração pode ser encontrado no Apêndice A, e os resultados obtidos estão apresentados na Tabela 1.

Tabela 1 – Quantidade de questões por fórum

Fórum	Quantidade de Questões
SO	102.819
SE	936
IT Security	1.166

Fonte: Autores

Como o SO teve 102.819 questões, foi preciso dividir em 3 consultas, como é explicado no Apêndice A.

[1] https://data.stackexchange.com/

4.2 Categorização das perguntas com LDA

Nesta seção, descreveremos o processo de categorização das perguntas utilizando o algoritmo Latent Dirichlet Allocation (LDA). Será explicado como foram executadas as etapas e seus resultados.

4.2.1 Análise Exploratória

Para obter uma visão inicial de quais seriam as palavras mais frequentes nos dados antes de executar o LDA, foram utilizadas algumas bibliotecas, como a **WordCloud** e a **matplotlib.pyplot**, ambas do Python. Essas ferramentas fornecem de forma visual as palavras mais comuns de um arquivo, onde as palavras que são exibidas de tamanho maior são as mais frequentes e as de menor tamanho são menos comuns.

Essa etapa de análise exploratória nos ajuda a compreender a distribuição das palavras em nosso conjunto de dados, identificando os termos mais relevantes e frequentes que podem estar relacionados aos tópicos que exploraremos posteriormente com o LDA.

A Figura 7 apresenta os resultados obtidos das 102.819 questões do Stack Overflow, a Figura 8 refere-se às 936 questões do Software Engineering, e por fim, a Figura 9 demonstra o resultado das 1.166 questões do IT Security.

Figura 7 – Fórum - Stack Overflow, Fonte: Autores

Figura 8 – Fórum - Software Engineering, Fonte: Autores

Figura 9 – Fórum - IT Security, Fonte: Autores

Por meio dessas figuras, podemos identificar as palavras comuns que aparecem com frequência nos fóruns, bem como aquelas que são mais prevalentes.

4.2.2 Carregando dados

O código desenvolvido para a implementação do LDA, no qual foram feitos todos os gráficos e imagens deste trabalho foi organizado em diferentes etapas e está em um arquivo chamado **topics.ipynb**, onde a extensão é ".ipynb"que representa *"Interactive Python Notebook"* (Notebook Python Interativo). O código completo pode ser consultado no repositório do Github [2].

Antes de explicar o código, a Tabela 2 mostra o *Schema* do *Stack Exchange Data*, com a descrição dos campos mais importantes para esse trabalho.

Tabela 2 – Esquema do Stack Exchange Data

Nome	Descrição
Id	Id da postagem
PostTypeId	Igual a 1 quando é pergunta e 2 quando é resposta
AcceptedAnswerId	Refere-se as respostas das postagens que foram aceitas pelo autor
CreationDate	Data que a postagem foi criada
ViewCount	Número total de visualizações por postagem
Body	Corpo da postagem
Title	Título da postagem
Tags	Tags do da postagem
AnswerCount	Número de respostas da postagem

4.2.2.1 Divisão dos dados

Devido à similaridade nos resultados dos fóruns, optou-se por consolidar todos os seus dados em um único arquivo para realizar uma análise abrangente. Para escrever o código, optamos por utilizar a biblioteca pandas do Python devido à sua flexibilidade e, em particular, pelo uso do Dataframe.

Nesse contexto, escolhemos a coluna *"Title"* como a mais relevante para a modelagem de tópicos. A razão para essa escolha reside no fato de que essa coluna reflete diretamente as consultas dos usuários, oferecendo uma visão direta do que está sendo pesquisado. A coluna *"Body"* também está disponível, porém, por conter explicações mais detalhadas das perguntas, optamos por focar exclusivamente no título para obter informações mais concisas sobre as dúvidas dos usuários.

[2] https://github.com/TCC-Emily-e-Vitor/Codigo-LDA

4.2.2.2 Escolha dos dados

Para determinar o número ideal de tópicos a serem gerados pelo LDA, executamos um script projetado para avaliar a coesão das palavras dentro dos tópicos, com o objetivo de identificar a configuração mais adequada. Exploramos um intervalo de 10 a 20 tópicos, aumentando um tópico de cada vez. Os resultados dessa análise são apresentados na Figura 10.

Figura 10 – Log dos tópicos gerados pelo LDA

Nesta análise, utilizamos uma biblioteca de processamento de linguagem natural em Python, na qual importamos o método *"stopwords"*. Estas *"stopwords"* são palavras consideradas de baixo valor na identificação de tópicos ou informações cruciais em um conjunto de documentos, devido à sua alta frequência e ocorrência generalizada na linguagem. Exemplos incluem palavras como *"it"*, *"on"* e outras similares.

Após incorporar as *"stopwords"* e executar o *script* para determinar a quantidade ideal de tópicos, obtivemos os resultados representados na Figura 11. O número de tópicos foi determinado pela pontuação de coerência foi 18, e seus respectivos tópicos estão detalhados na Figura 12.

Entretanto, observamos que os tópicos gerados inicialmente continham palavras de pouca relevância ou termos com múltiplos significados. Exemplos dessas palavras incluíam termos como "6", "5"e similares.

Em resposta a isso, tomamos medidas adicionais durante a execução do código. Foi incorporado um conjunto adicional de palavras para aprimorar o processo de filtragem, resultando em uma lista final que inclui as seguintes palavras: *'using', 'working', 'like', 'could', 'set', 'change', 'value', 'get', '2', 'load', 'getting', 'new', 'use', 'filter', '6', '5', '1', '3'*.

Dessa forma, o código foi executado várias vezes até que se obtivesse um resultado satisfatório. O número ideal de tópicos foi determinado como sendo 16, conforme

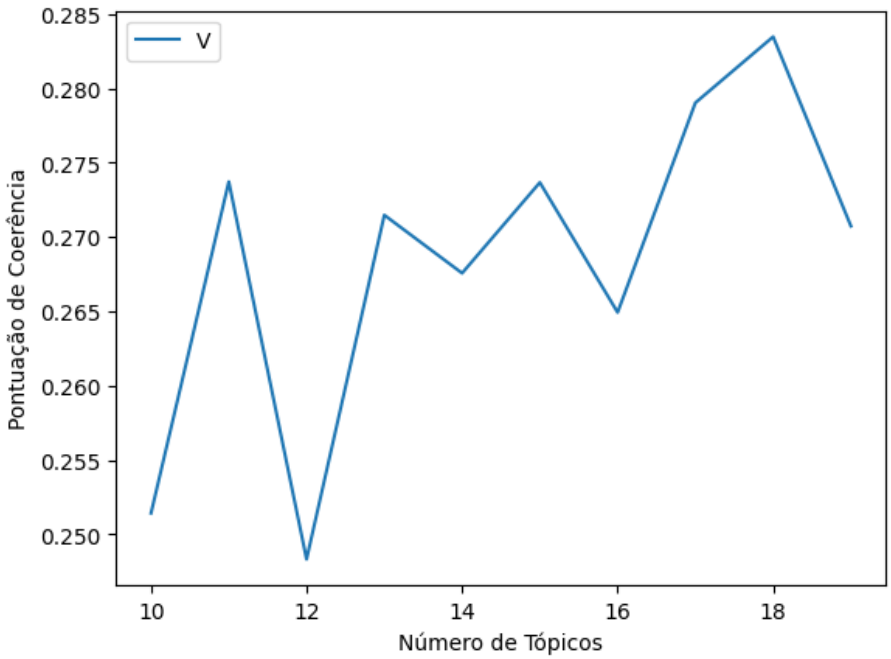

Figura 11 – Quantidade de tópicos sugeridos pelo gensim antes

demonstrado na Figura 13, e os respectivos tópicos podem ser visualizados na Figura 14.

4.2.2.3 Categorização dos Tópicos

Para cumprir com o objetivo de categorizar as perguntas extraídas dos fóruns em grupos maiores de mesmo assunto, optamos por seguir algumas abordagens a fim de extrair a que tivesse o melhor resultado. Sendo assim, uma vez separadas as principais palavras que definem cada tópico, fazendo uso do LDA, precisamos agora definir qual assunto melhor representa essas palvras agrupadas em um mesmo tópico.

Inicialmente fizemos uso do Word2Vec, modelo de aprendizado de máquina utilizado para a representação vetorial de palavras em um espaço contínuo. Dessa forma, o modelo foi utilizado para encontrar palavras e semelhanças entre os tópicos gerados pelo LDA, tendo como resultado um outro vetor de palavras que represente o 'centroid' desse *input* do LDA.

Dessa forma, o Word2Vec nos forneceu um conjunto de palavras para cada tópico que representa as relações e similaridades de contexto entre as palavras fornecidas pelo LDA em cada tópico.

Após isso, foi utilizado o modelo GPT-3 (Generative Pre-trained Transformer 3) para realização da tarefa de classificação de texto, no qual ele é capaz de classificar em diferentes categorias ou tópicos um conjunto de texto. Dessa forma, utilizamos esse modelo para gerar as nossas categorias centrais, e o resultado obtido pode ser visto na Figura 15

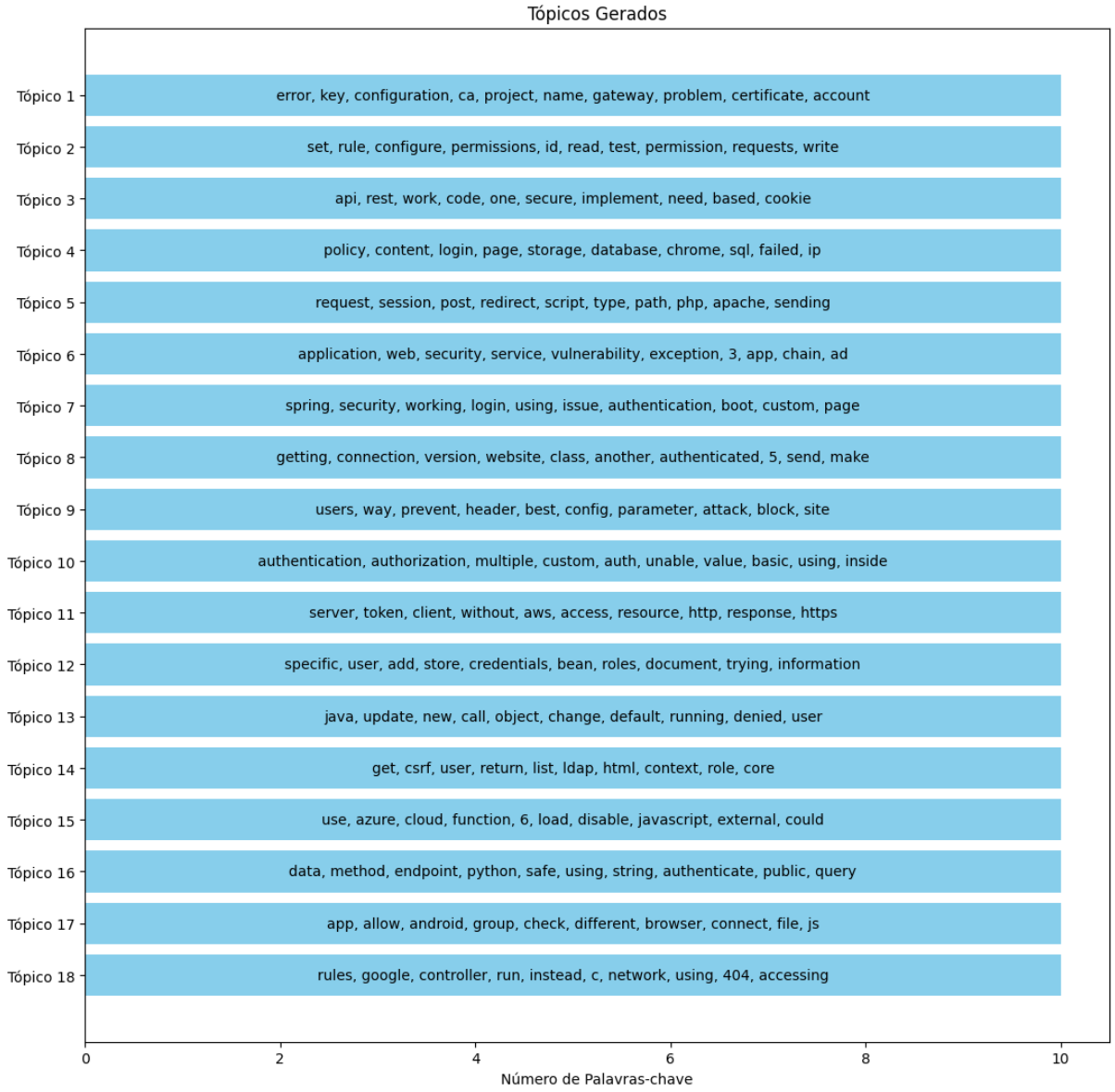

Figura 12 – Tópicos gerados pelo LDA antes

4.2.3 Resultados

No contexto desse processamento de dados e divisão das perguntas em diferentes categorias, foram gerados gráficos para a visualização dos resultados e para que a divisão das perguntas entre os tópicos e os anos sejam melhor avaliadas.

A primeira visualização desenvolvida foi a distribuição de perguntas em cada tópico criado pelo LDA, assim podemos ter uma noção de quais são os tópicos com mais perguntas. Conforme ilustrado na Figura 16, podemos verificar que as categorias: "Seguranças em Aplicações PHP", "Segurança de Comunicações Web"e "Problemas de Acesso e Segurança em APIs"são as categorias que obtiveram mais dúvidas durante os anos de 2008 até 2023.

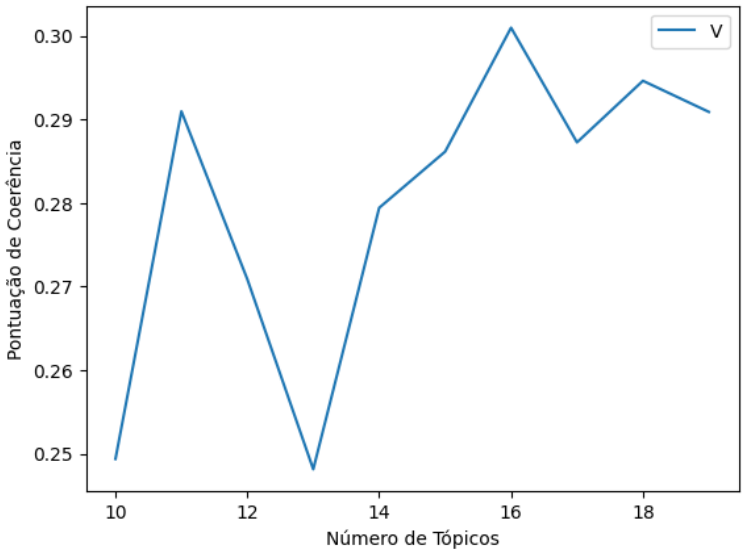

Figura 13 – Quantidade de tópicos sugeridos pelo gensim depois

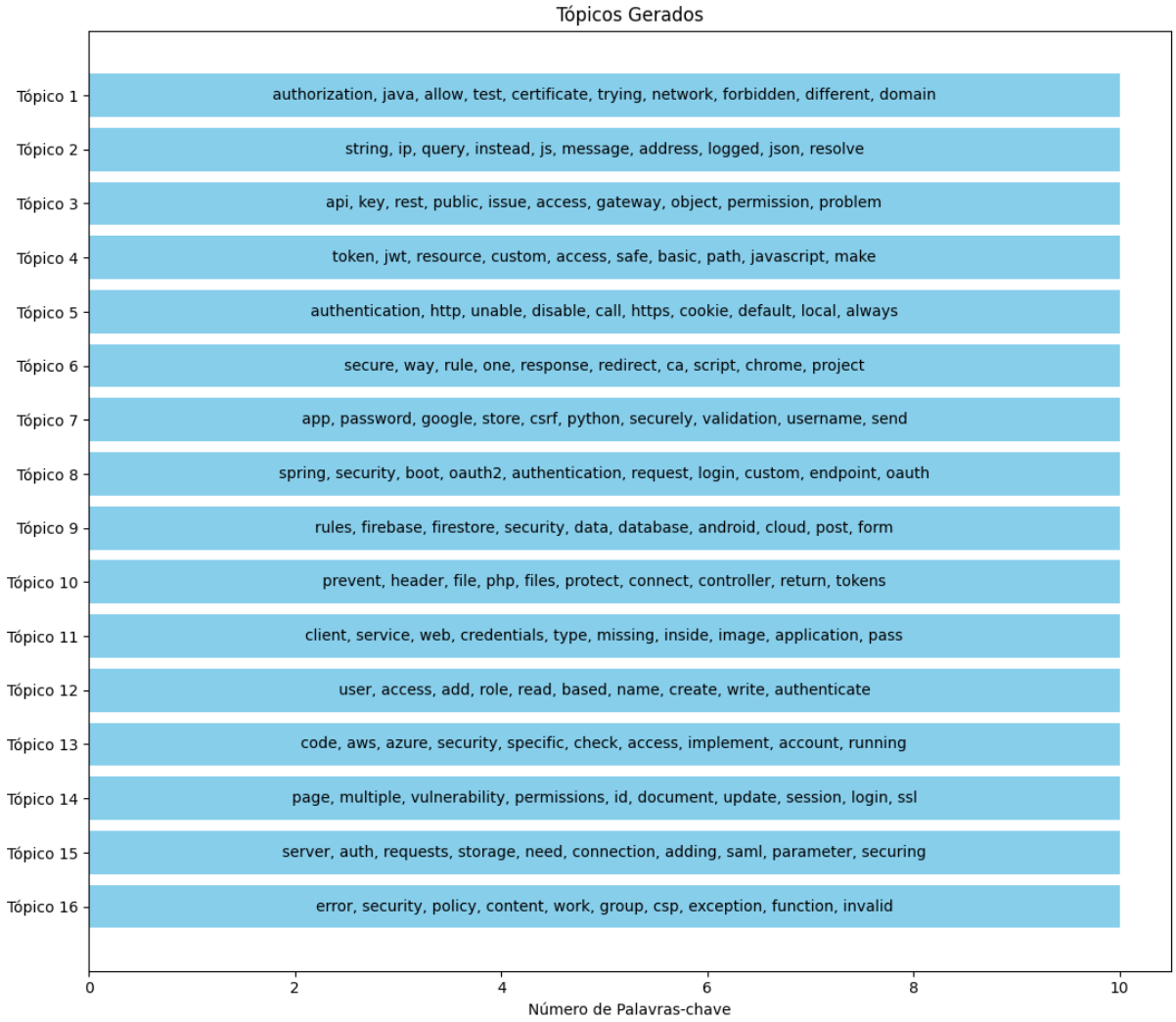

Figura 14 – Tópicos gerados pelo LDA depois

Tópico 1: Gerenciamento de Dados em Nuvem
Tópico 2: Autenticação e Segurança de Aplicações
Tópico 3: Integração de Serviços e Autenticação
Tópico 4: Segurança de Senhas e Armazenamento de Credenciais
Tópico 5: Manipulação de Dados em JavaScript
Tópico 6: Segurança de Conexões e Autenticação de Servidor
Tópico 7: Segurança de Contas e Acesso a Serviços em Nuvem
Tópico 8: Segurança em Aplicações PHP
Tópico 9: Segurança de Comunicações Web
Tópico 10: Autorização em Java e Certificados
Tópico 11: Gerenciamento de Permissões e Autenticação em Documentos
Tópico 12: Autenticação Baseada em Funções de Usuário
Tópico 13: Segurança e Acesso em JavaScript
Tópico 14: Certificados e Redirecionamento em Scripts
Tópico 15: Erros de Segurança e Políticas de Conteúdo
Tópico 16: Problemas de Acesso e Segurança em APIs

Figura 15 – Categorias criadas pelo modelo GPT-3

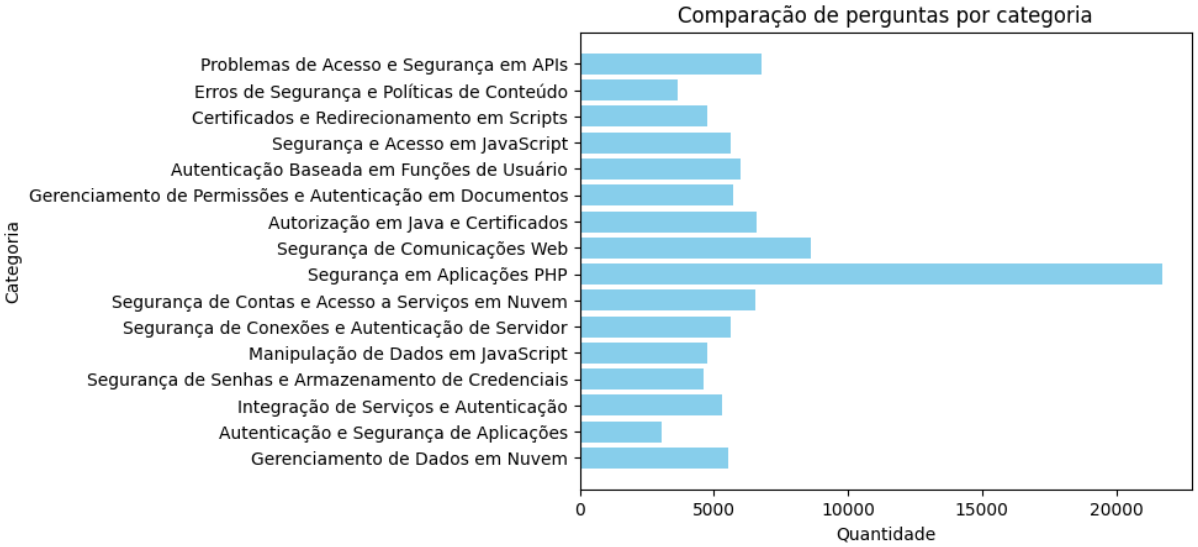

Figura 16 – Quantidade de Perguntas por Categoria

Em relação à quantidade de perguntas feitas nos três fóruns ao passar dos anos, podemos perceber que o gráfico possui um comportamento parabólico, começando com um número baixo, tendo o seu auge nos anos de 2015 e 2016 e depois disso entrou em uma tendência decrescente, como pode ser visto na Figura 17.

Pode-se perceber pelo gráfico que a quantidade de perguntas nos últimos anos começou a cair, podendo ser um sinal de que o conhecimento na área de Segurança de Software esteja mais acessível e mais consolidado entre a comunidade de software, uma vez que o número de questões no ano de 2023, apesar de ainda não finalizado no momento desse estudo, esteja perto da metade da quantidade de questões no ano de 2015.

O Apêndice B apresenta os gráficos ano a ano, sendo possível visualizar o pro-

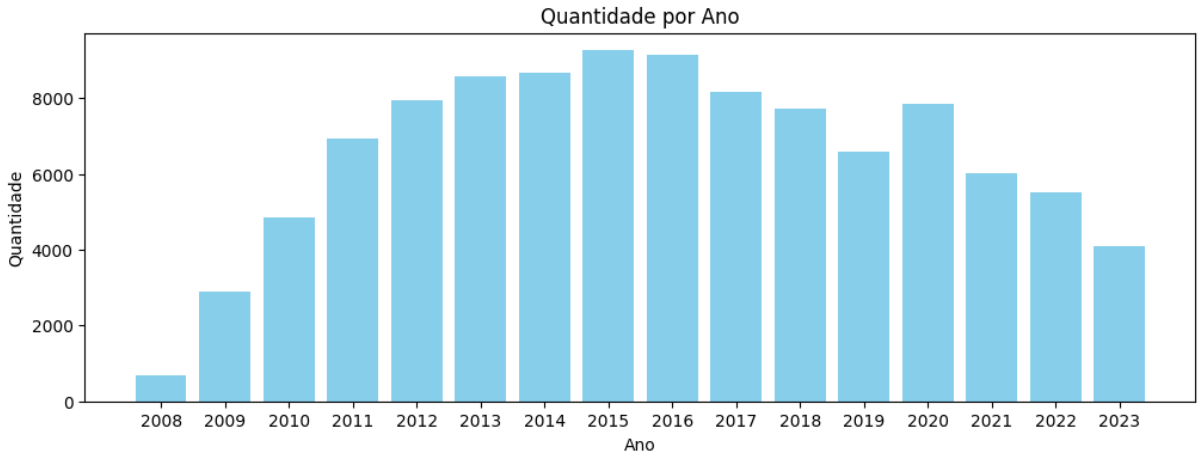

Figura 17 – Quantidade de Perguntas por Ano

gresso individual de cada categoria ao longo do tempo. Observa-se que no ano de 2008 "Segurança de Software"era uma temática pouco discutida nos fóruns pesquisados e que nos últimos anos as categorias como um todo tiveram uma redução na quantidade de perguntas realizadas, assim como indicado pela Figura 17 de forma sumarizada.

Além disso, podemos destacar a quantidade significativa de perguntas para a categoria "Segurança em Aplicações PHP", que a partir de 2012 passou a se destacar em comparação com as outras categorias, mantendo a liderança em quantidade de perguntas em todos os outros anos.

E por fim, a Figura 18 apresenta um gráfico da quantidade de perguntas por categoria criadas no decorrer dos anos.

4.3 Análise das Tags relacionadas às perguntas

Cada tópico gerado inclui um conjunto de *tags* relacionadas. Para analisar a correlação entre essas *tags*, realizamos uma série de testes para determinar a maneira mais eficaz de apresentar esses dados.

No processo de geração das *tags* mais frequentes, os códigos excluíram intencionalmente palavras que continham a sequência 'security' em seu interior. Isso ocorreu porque as perguntas já haviam passado por uma filtragem que incluiu a *tag* 'security'. Por exemplo, os termos 'security-content' e 'security' estão entre os descartados.

Para efetuar a filtragem das *tags*, incorporamos um código que remove a marcação HTML, uma vez que originalmente as *tags* estavam no formato <tag1><tag2>. Esse filtro converteu as tags para o formato de *array*, tornando-as assim: [tag1, tag2], pois fica mais simples de se trabalhar com os dados dessa forma no código.

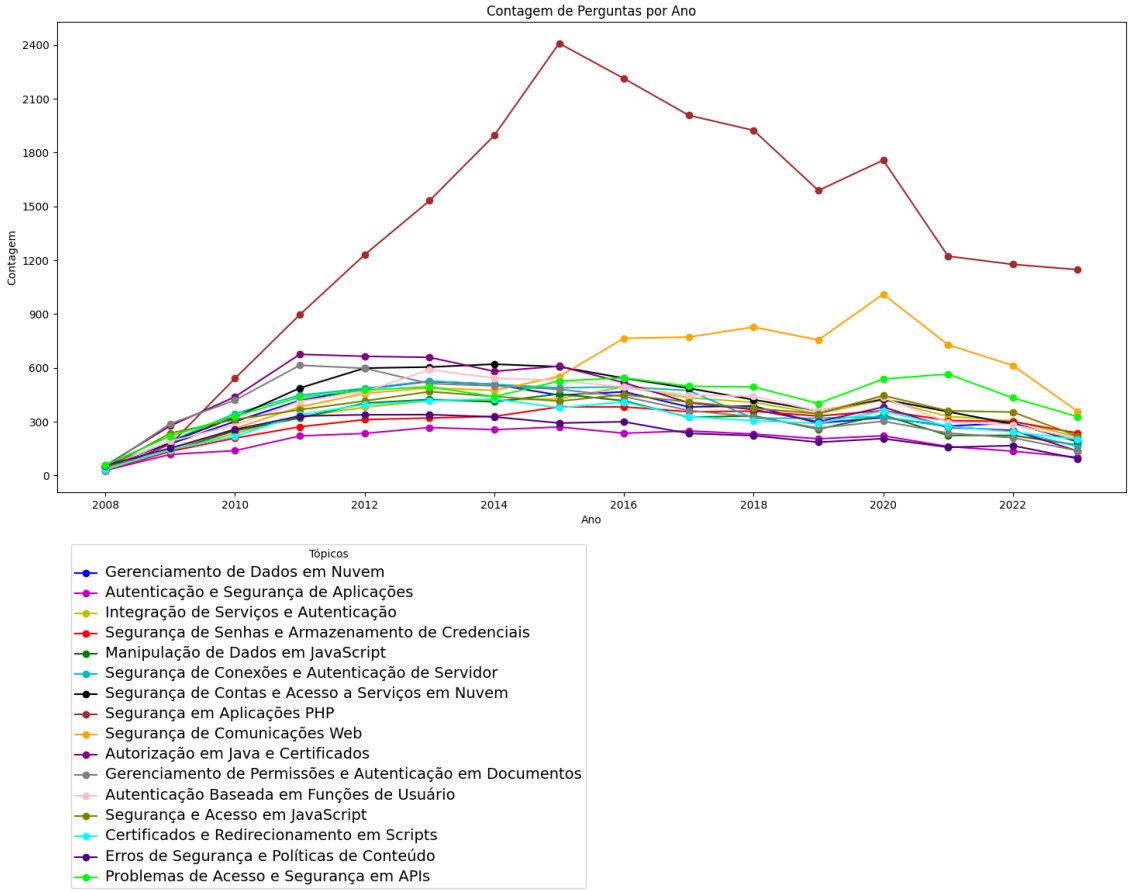

Figura 18 – Quantidade de Perguntas por Ano de Cada Categoria

4.3.1 Resultados

Foram criados dois tipos de representações visuais para este fim. O primeiro consiste em um gráfico que exibe as *tags* gerais presentes nos três fóruns, permitindo uma avaliação das métricas que relacionam os tópicos entre os fóruns. O segundo método de análise envolve a criação de uma tabela que relaciona cada fórum às *tags* específicas presentes nos tópicos gerados.

Como nos gráficos anteriores, os gráficos abaixo são da biblioteca do Python chamada matplotlib [3]. Essa biblioteca facilita a visualização dos dados.

4.3.1.1 Tags mais utilizadas geral

A escolha de exibir 16 *tags* foi baseada na percepção de que esse valor proporciona uma representação significativa delas. No qual, é um valor que reflete uma seleção abrangente de resultados, beneficiando-se da diversidade de *tags* disponíveis.

As Figuras 19 e 20 ilustram a frequência de ocorrência de cada *tag* nos tópicos, somando as contribuições dos três fóruns. O motivo de dividir em duas imagens está

[3] https://matplotlib.org/stable/gallery/index.html

vinculada à grande quantidade de dados disponíveis SO, o que resulta em uma contagem mais extensa. Dessa forma, optou-se por separar em duas imagens para proporcionar uma visão mais detalhada dos resultados em cada fórum.

Pelos resultados, é possível perceber que as *tags* mais utilizadas são as baseadas na linguagem Java, pois o top 3 é "java", "spring"e "spring-boot". Java é uma das linguagens de programação mais amplamente utilizadas, com aplicações que abrangem diversos domínios, incluindo desenvolvimento móvel, criação de jogos, aplicativos de *desktop* e muito mais. Portanto, não é surpreendente que ela tenha ocupado o primeiro lugar nas *tags*.

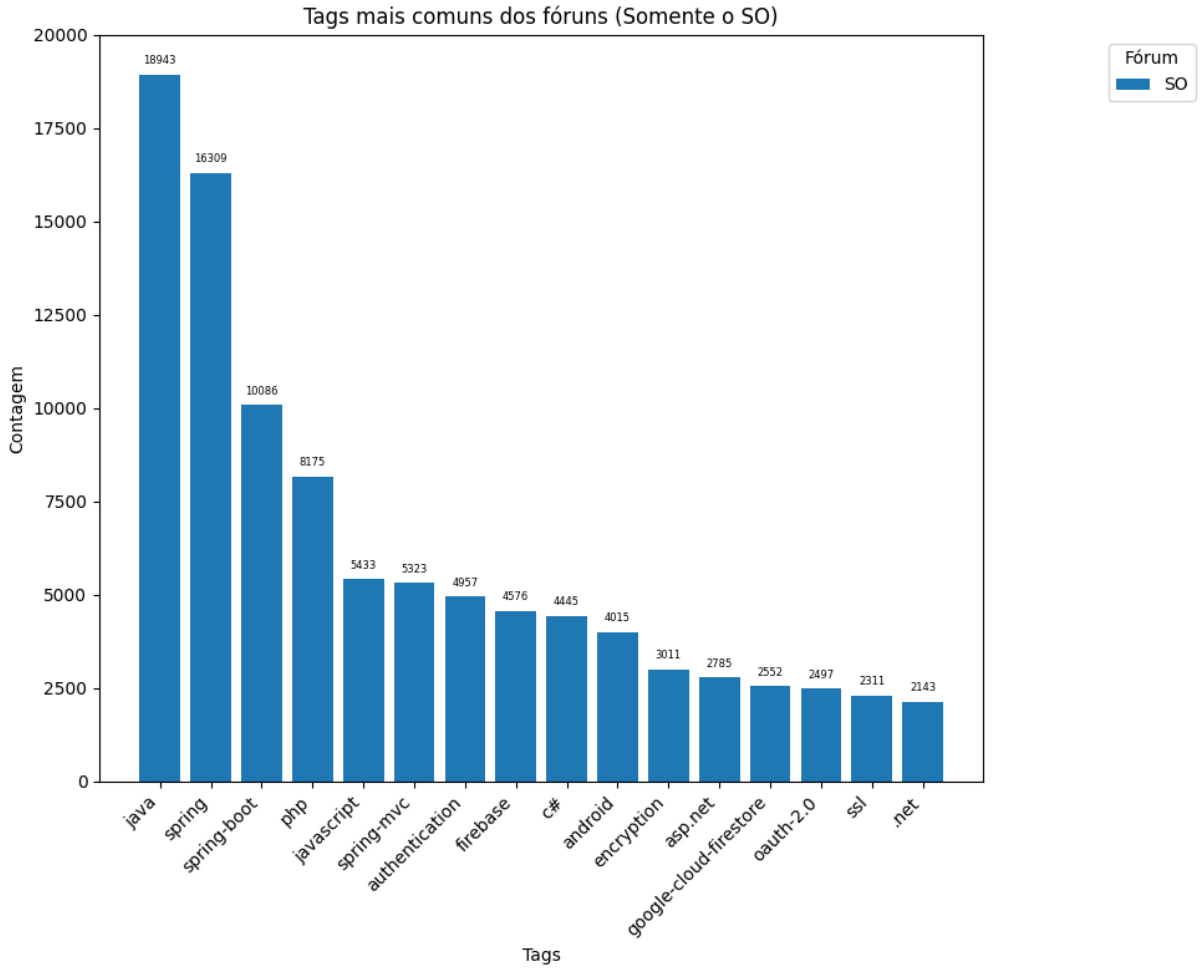

Figura 19 – Frequência das tags do SO

4.3.1.2 Tags mais utilizadas de cada fórum

A Tabela 3 mostra as cinco *tags* mais frequentes nos fóruns de acordo com cada categoria.

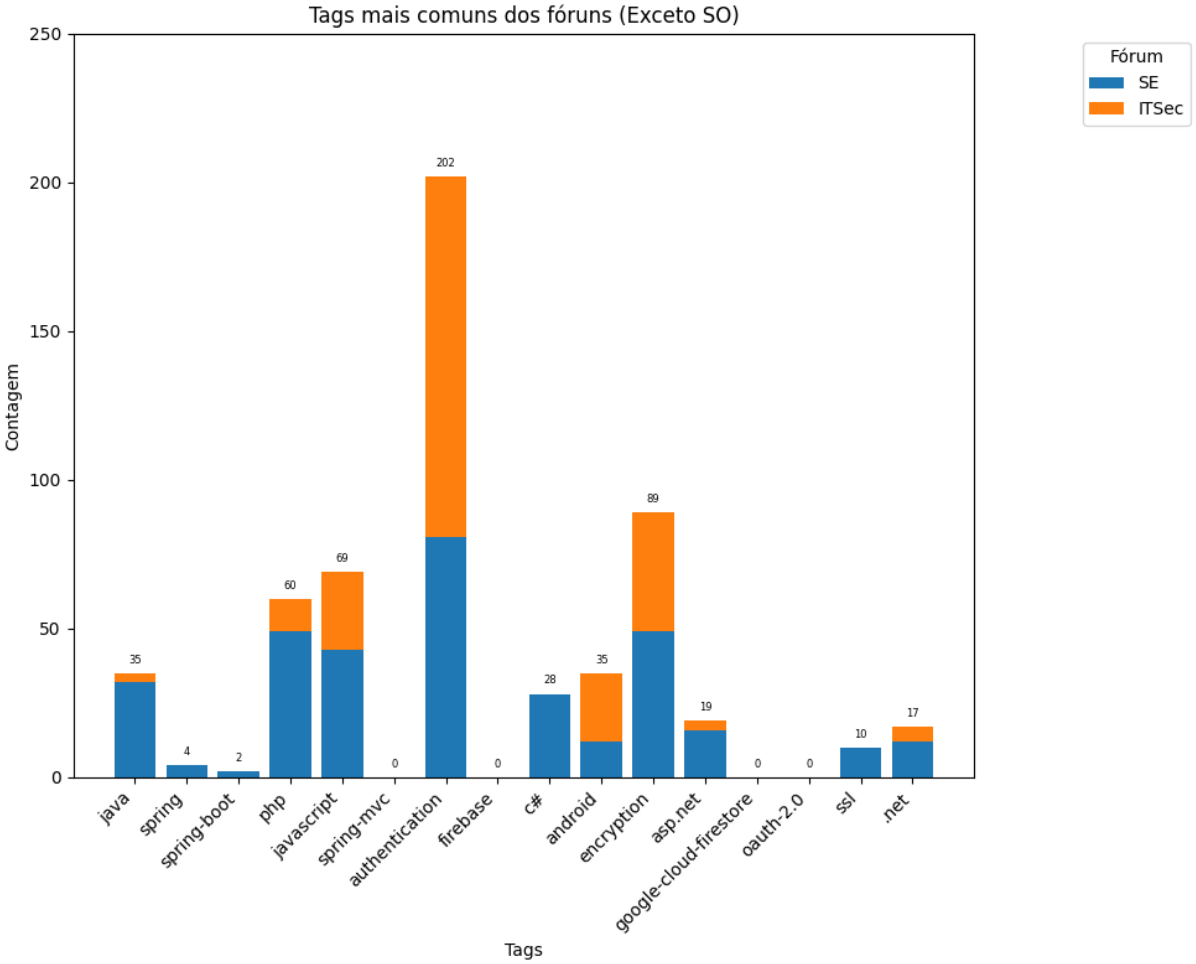

Figura 20 – Frequência das tags sem o SO

Tabela 3 – Tags mais utilizadas de cada fórum

SO	SE	ITSec
#1 Gerenciamento de Dados em Nuvem		
java: 1550 vezes	authentication: 7 vezes	network: 7 vezes
spring: 501 vezes	authorization: 5 vezes	waf: 5 vezes
spring-boot: 334 vezes	ssl: 4 vezes	web-application: 5 vezes
c#: 301 vezes	microservices: 4 vezes	passwords: 4 vezes
ssl: 298 vezes	rest: 3 vezes	malware: 4 vezes
#2 Autenticação e Segurança de Aplicações		
java: 423 vezes	passwords: 5 vezes	email: 7 vezes
php: 305 vezes	java: 3 vezes	passwords: 6 vezes
javascript: 228 vezes	email: 3 vezes	ip: 6 vezes
spring: 219 vezes	hacking: 2 vezes	authentication: 5 vezes
c#: 141 vezes	validation: 2 vezes	privacy: 5 vezes

Continua na próxima página

#3 Integração de Serviços e Autenticação		
java: 838 vezes	api: 21 vezes	encryption: 6 vezes
rest: 427 vezes	rest: 18 vezes	api: 5 vezes
encryption: 409 vezes	encryption: 9 vezes	authentication: 5 vezes
spring: 372 vezes	authentication: 9 vezes	web-application: 4 vezes
api: 334 vezes	architecture: 8 vezes	email: 4 vezes
#4 Segurança de Senhas e Armazenamento de Credenciais		
java: 756 vezes	authentication: 8 vezes	authentication: 5 vezes
spring: 665 vezes	javascript: 6 vezes	jwt: 4 vezes
spring-boot: 500 vezes	rest: 4 vezes	privacy: 3 vezes
jwt: 483 vezes	design: 3 vezes	malware: 3 vezes
javascript: 436 vezes	java: 3 vezes	xss: 3 vezes
#5 Manipulação de Dados em Javascript		
java: 734 vezes	authentication: 11 vezes	authentication: 13 vezes
spring: 533 vezes	http: 3 vezes	multi-factor: 7 vezes
authentication: 439 vezes	https: 2 vezes	web-application: 6 vezes
spring-boot: 344 vezes	web-development: 2 vezes	xss: 5 vezes
https: 331 vezes	algorithms: 2 vezes	cookies: 4 vezes
#6 Segurança de Conexões e Autenticação de Servidor		
php: 748 vezes	c#: 6 vezes	passwords: 14 vezes
java: 639 vezes	javascript: 5 vezes	authentication: 6 vezes
grails: 521 vezes	authentication: 5 vezes	p-m: 6 vezes
spring: 450 vezes	encryption: 5 vezes	apache: 5 vezes
javascript: 360 vezes	php: 4 vezes	web-application: 5 vezes
#7 Segurança de Contas e Acesso a Serviços em Nuvem		
java: 750 vezes	passwords: 13 vezes	passwords: 27 vezes
encryption: 635 vezes	web-applications: 10 vezes	authentication: 21 vezes
android: 576 vezes	encryption: 9 vezes	google: 19 vezes
php: 473 vezes	hashing: 7 vezes	email: 19 vezes
passwords: 454 vezes	database: 7 vezes	multi-factor: 15 vezes
#8 Segurança em Aplicações PHP		
spring: 10296 vezes	authentication: 6 vezes	authentication: 5 vezes
java: 7925 vezes	web-application: 3 vezes	design: 5 vezes
spring-boot: 6357 vezes	webserver: 3 vezes	php: 3 vezes
spring-mvc: 3308 vezes	phone: 3 vezes	authorization: 3 vezes
oauth-2.0: 1298 vezes	passwords: 2 vezes	open-source: 2 vezes
#9 Segurança de Comunicações Web		

Continua na próxima página

firebase: 3142 vezes	database: 13 vezes	passwords: 8 vezes
g-c-f: 1860 vezes	passwords: 11 vezes	android: 7 vezes
f-r-d: 1362 vezes	encryption: 8 vezes	web-application: 6 vezes
android: 1147 vezes	web-development: 7 vezes	databases: 5 vezes
php: 641 vezes	architecture: 6 vezes	authentication: 4 vezes
#10 Autorização em Java e Certificados		
php: 2173 vezes	javascript: 8 vezes	xss: 7 vezes
mysql: 576 vezes	php: 8 vezes	sql-injection: 7 vezes
javascript: 464 vezes	sql-injection: 5 vezes	web-application: 6 vezes
java: 462 vezes	web-development: 4 vezes	privacy: 5 vezes
xss: 413 vezes	database: 3 vezes	authentication: 4 vezes
#11 Gerenciamento de Permissões e Autenticação em Documentos		
wcf: 1141 vezes	passwords: 9 vezes	passwords: 9 vezes
java: 646 vezes	authentication: 7 vezes	authentication: 6 vezes
c#: 591 vezes	web-development: 6 vezes	web-application: 6 vezes
web-services: 442 vezes	rest: 6 vezes	tls: 3 vezes
authentication: 348 vezes	architecture: 6 vezes	credentials: 3 vezes
#12 Autenticação Baseada em Funções de Usuário		
java: 895 vezes	login: 5 vezes	web-application: 9 vezes
spring: 644 vezes	database-design: 5 vezes	authentication: 9 vezes
php: 535 vezes	authentication: 5 vezes	passwords: 7 vezes
authentication: 386 vezes	database: 5 vezes	encryption: 6 vezes
spring-boot: 380 vezes	data: 4 vezes	access-control: 4 vezes
#13 Segurança e Acesso em JavaScript		
java: 823 vezes	open-source: 11 vezes	authentication: 12 vezes
a-w-s: 378 vezes	web-development: 8 vezes	passwords: 8 vezes
php: 360 vezes	web-applications: 7 vezes	windows: 6 vezes
c#: 332 vezes	python: 4 vezes	email: 6 vezes
javascript: 273 vezes	web-services: 4 vezes	web-application: 6 vezes
#14 Certificados e Redirecionamento em Scripts		
java: 626 vezes	authentication: 8 vezes	php: 6 vezes
spring: 495 vezes	web-application: 5 vezes	javascript: 4 vezes
php: 400 vezes	cookies: 4 vezes	database: 3 vezes
ssl: 315 vezes	passwords: 4 vezes	web-development: 3 vezes
spring-boot: 274 vezes	xss: 3 vezes	session: 3 vezes
#15 Erros de Segurança e Políticas de Conteúdo		
java: 461 vezes	web-applications: 4 vezes	authentication: 6 vezes

Continua na próxima página

php: 350 vezes	web-development: 4 vezes	encryption: 4 vezes
sql-server: 272 vezes	javascript: 3 vezes	webserver: 3 vezes
encryption: 242 vezes	web-services: 3 vezes	xss: 3 vezes
spring: 232 vezes	database: 3 vezes	multi-factor: 3 vezes
#16 Problemas de Acesso e Segurança em APIs		
java: 844 vezes	php: 5 vezes	xss: 26 vezes
javascript: 676 vezes	architecture: 5 vezes	web-application: 13 vezes
spring: 494 vezes	p-p: 4 vezes	http: 8 vezes
php: 442 vezes	browser: 4 vezes	web-browser: 8 vezes
c#: 342 vezes	web-development: 4 vezes	authentication: 7 vezes

Alguns casos ocorreu da *tag* ser muito longa, então foi adicionado a abreviação que é encontrada na Tabela 4, para não dificultar a visualização.

Tabela 4 – Abreviação das Tags

Tag	Abreviação
password-management	p-m
google-cloud-firestore	g-c-f
firebase-realtime-database	f-r-d
amazon-web-services	a-w-s
programming-practices	p-p

4.4 Identificação das perguntas mais difíceis

Essa etapa mostra o tempo que as perguntas relacionadas com as categorias demoram para serem respondidas, essa métrica é interessante para mostrar os tópicos com mais dificuldades de resposta. A Tabela 5 mostra os aspectos propostos nesse trabalho.

Tabela 5 – Média de Números de Visualizações (MNV), Porcentagem de Perguntas Com Respostas (PPCR), Média de Número de Respostas (MNR), Porcentagem de Perguntas Sem Respostas Aceitas (PPSRA) e Média de Tempo para Receber Respostas (em dias) (MTR).

Categoria	**MNV**	**PPCR**	**MNR**	**PPSRA**	**MTR**
Gerenciamento de Dados em Nuvem	3636	81.94%	1.37	54.47%	18.39
Autenticação e Segurança de Aplicações	3312	84.89%	1.57	52.43%	19.55
Integração de Serviços e Autenticação	2879	83.36%	1.40	53.00%	24.14
Segurança de Senhas e Armazenamento de Credenciais	3163	82.37%	1.38	53.94%	20.69
Manipulação de Dados em Javascript	4767	82.94%	1.47	53.53%	20.47
Segurança de Conexões e Autenticação de Servidor	2927	85.34%	1.56	50.54%	18.27
Segurança de Contas e Acesso a Serviços em Nuvem	3227	83.57%	1.54	51.35%	21.50
Segurança em Aplicações PHP	3204	79.83%	1.20	57.21%	21.49
Segurança de Comunicações Web	1769	83.85%	1.33	50.64%	11.79
Autorização em Java e Certificados	3416	86.80%	1.81	47.18%	14.10
Gerenciamento de Permissões e Autenticação em Documentos	3202	84.06%	1.39	51.50%	21.36
Autenticação Baseada em Funções de Usuário	2553	84.91%	1.42	50.82%	13.60
Segurança e Acesso em JavaScript	2837	83.51%	1.46	52.54%	26.06
Certificados e Redirecionamento em Scripts	2433	83.39%	1.40	53.00%	21.23
Erros de Segurança e Políticas de Conteúdo	3340	84.11%	1.52	51.46%	19.38
Problemas de Acesso e Segurança em APIs	3829	82.31%	1.41	53.81%	15.26

4.4.1 Implicações

Diante dos dados da tabela, destacam-se conclusões relevantes acerca de categorias específicas que merecem atenção. A análise a seguir detalhará as categorias de maior destaque, fornecendo uma explicação com foco nos números que evidenciam desafios específicos relacionados ao tema em questão.

A primeira categoria que demanda especial atenção é "Autorização em Java e Certificados", destacando-se por dados extremamente positivos. Esta categoria não apenas

ocupa como a quarta com o maior número de visualizações (3416), mas também é a mais elevada porcentagem de respostas por pergunta (86.80%). Mesmo ocupando essa posição em quantidade de perguntas, ela se destaca como a categoria com a média mais alta de respostas por pergunta (1.81). Além disso, ocupa o terceiro lugar no menor tempo médio de resposta, com uma média de 14.10 dias por pergunta. Notavelmente, apresenta o menor percentual de perguntas sem respostas aceitas (47.18%), refletindo sua relevância no cenário atual. Estes indicadores evidenciam que o tópico tem sido amplamente discutido pelos usuários dos fóruns, sendo caracterizado por uma expressiva participação e um notável índice de perguntas respondidas e aceitas. Esses dados deixam bastante evidente que esta categoria é a que apresentou menor dificuldade dos usuários, se comparado com as outras, sendo líder nos dados que possuem bastante peso para classificar uma categoria como difícil ou não, como o menor número percentual de perguntas sem resposta aceita e a terceira categoria com o menor tempo de respostas.

Por outro lado, a categoria "Segurança em Aplicações PHP"apresentou o maior percentual de perguntas sem respostas aceitas, apresentando 57.21%, um número bem expressivo se comparado com as outras categorias, sendo 2.74% maior que a segunda categoria e 10.03% maior se comparado com a última categoria. Além disso, essa categoria apresenta a menor média de visualizações por pergunta, é a quarta categoria com o maior tempo de resposta e a categoria que possui a menor porcentagem de perguntas com respostas. Unindo todos esses dados, ao fator de que essa categoria possui uma visualização média de 3204 por pergunta, que não chega a ser um número baixo se comparado com as outras categorias, fica evidente que existe uma certa dificuldade no assunto, uma vez que as perguntas estão sendo vistas, ou buscadas pelos usuários, mas ainda assim não possuem uma quantidade considerável de respostas e respostas aceitas, o que demonstra uma certa dificuldade por parte dos usuários, podendo considerá-la até como a categoria mais difícil dentre as outras apresentadas.

Assim como a última categoria comentada, "Gerenciamento de Dados em Nuvem"apresenta dados muito similares, possui a segunda maior porcentagem de perguntas sem respostas aceitas (54.47%), a terceira menor média de número de respostas por pergunta, a segunda menor porcentagem de perguntas com respostas, e uma média de 18.39 dias para receber uma resposta. Além disso, possui um número de visualizações um pouco maior do que a última categoria, obtendo uma média de 3636 visualizações por pergunta. Podemos perceber que as análises e os dados são muito similares à categoria "Segurança em Aplicações PHP", portanto pode-se considerar que esta é a segunda categoria mais difícil, comparado com as outras.

Por fim, temos a categoria "Segurança de Senhas e Armazenamento de Credenciais", que obtém a terceira maior porcentagem de perguntas sem respostas aceitas (53.94%), a quarta menor média de respostas por pergunta (1.38), a quarta com menor percentual de

perguntas com respostas (82.37%) e um tempo médio para obter respostas de 20.69 dias. A sua média de visualização segue o mesmo padrão das últimas duas, obtendo 3163 por pergunta, e portanto podemos considerar que esta seria a terceira categoria mais difícil dentre as apresentadas.

4.5 Comparação dos Estudos

Após extração dos resultados, podemos comparar o nosso estudo e o que foi obtido com os resultados do artigo Rahman (2016). Como citado anteriormente, o estudo de Rahman também foi realizado na área de 'Segurança de Software', em 2016, e um dos pontos principais de destaque é a semelhança em alguns tópicos, mesmo com nomenclaturas diferentes, como mostra a Tabela 6. Portanto, passados sete anos podemos perceber que algumas categorias ainda são bastante utilizadas pelos usuário dos fóruns.

Tabela 6 – Comparação das tags entre as duas pesquisas

Tags (RAHMAN, 2016)	**Tags deste estudo**
php, encryption, authentication, java, ssl, passwords, xss, cryptography, javascript, https, hash, spring, android, login	java, spring, spring-boot, php, javascript, spring-mvc, authentication, firebase, c#, android, encryption, asp.net, google-cloud-firestore, oauth-2.0, ssl, .net

Algumas categorias são mais gerais, o que podem explicar a aparição nos dois estudos, mas existem categorias nichadas em alguns dos principais tópicos de 'Segurança de Software' que conhecemos até hoje. Sendo assim, as categorias que podemos destacar que aparecem nos dois estudos são categorias com o assunto principal sendo: 'Segurança de Senhas' (Tópico 4), 'Autenticação e Segurança de Aplicações' (Tópico 2), 'Segurança de Comunicações Web (Tópico 9), 'Autorizações em Java' (Tópico 10). No estudo de Rahman (2016) essas categorias foram nomeadas respectivamente da seguinte forma: *'Password Storage'*, *'Authentication'*, *'Web Security'* e *'Java Implementation'*.

Por outro lado, podemos perceber o desuso de uma categoria do artigo do Rahman (2016), assim como a aparição de novas categorias neste trabalho. No caso do estudo de Rahman (2016) a categoria 'Grails Framework (Implementation)' não possui nenhuma similaridade em tags e em assunto principal com nenhum dos nossos tópicos. Grails é um framework para desenvolvimento web para a linguagem de programação Groovy que acabou perdendo popularidade devido ao crescimento de outras tecnologias como Spring Boot, na qual foi uma *tag* bem frequente em nosso estudo, Micronaut e Quarkus. No caso do nossa pesquisa, podemos perceber que algumas categorias com novas *tags* e assuntos principais aparecem em meio aos tópicos, como destaque podemos comentar sobre 'Ma-

nipulação de Dados em Javascript' e 'Segurança e Acesso em JavaScript'. Os dois tópicos giram em torno de Javascript, que ganhou muita popularidade do ano de 2016 para o ano atual.

Fazendo uma análise mais detalhada das categorias presentes em ambas as pesquisas, podemos examinar, nas Figuras de 21 a 36, como essas categorias se comportaram ao longo dos anos que as separam, dado que a pesquisa em questão leva em consideração questões feitas a partir de 2008, e portanto essa aparição podem representar que o assunto foi muito discutido em anos passados, e não necessariamente nos anos após ao estudo de Rahman (2016). Começando pela categoria 'Segurança de Senhas' (Tópico 4) pode-se perceber que ela manteve em todos os anos uma média muito próxima de 400 perguntas, o que indica que não é um assunto que ficou no passado, mas que até os dias atuais é um dos focos de discussão dentro dos fóruns, explicando o por quê de estar presente nos dois estudos.

'Autenticação e Segurança de Aplicações' (Tópico 2) vem tendo o seu número de perguntas por ano diminuindo desde 2020, sendo a categoria com menos perguntas no ano atual, o que nos mostra que nos últimos anos teve uma quantidade considerável de perguntas, para que entrasse como tópico nos dois estudos, mas seguindo o resultado dos últimos anos, a tendência é que continue diminuindo. Um ponto importante a se destacar é que esse tópico é um assunto geral, e com o avanço da informação e a melhoria dos fóruns, é comum que cada vez mais as perguntas sejam mais específicas, assim como os problemas enfrentados. A base teórica e a quantidade de materiais generalizados sobre 'Autenticação e Segurança de Aplicações' é muito grande nos tempos atuais, não sendo tão necessário e comum os usuários buscarem esse tipo de informações nos fóruns, e sim problemas mais específicos.

No caso de 'Segurança de Comunicações Web', é perceptível que a categoria não tinha tanta expressão, se comparada com as outras, até o ano de 2015, que então começou a assumir a posição de segunda categoria com mais perguntas dentre os tópicos selecionados, muito pela popularização do assunto e o crescimento, tanto de tecnologias para o desenvolvimento web, quanto no interesse e demanda do mercado de software.

Outra comparação que podemos fazer, e que acompanha o que foi comentado anteriormente, é sobre a utilização das tags nas perguntas. Segundo o estudo de Rahman (2016) as *tags* que mais apareceram nas questões utilizadas no estudo foram: PHP, Java, Javascript e *Encryption*. Apesar de 'Javascript' aparecer como uma das *tags* mais frenquentes, ela tem uma frequência de quase um terço da primeira, o que mostra a disparidade mesmo entre as *tags* mais utilizadas. No escopo da nossa pesquisa, vimos que as mais frequentes são: 'java', 'spring', 'spring boot', 'php' e 'javascript'.

Sendo assim, Spring e Spring Boot foram duas *tags* que passaram na frente da tag 'Java' se comparado com o estudo de Rahman (2016). Por outro lado, a *tag 'Encryption'*

passou da quarta posição para a décima primeira em nosso estudo, sendo ultrapassada por alguns assuntos em ascenção como 'firebase' e 'android' representando segurança mobile, por exemplo.

5 Conclusão

Neste trabalho foi apresentado um estudo sobre segurança de software, onde foi discutido o que os desenvolvedores estão pesquisando sobre o assunto. Com o auxílio de outros artigos, foi possível fazer uma linha do tempo com tópicos sobre as dificuldades enfrentadas pela comunidade e com isto, realizar um levantamento desses dados. Foi possível desenvolver esse estudo com o suporte dos fóruns Stack Overflow, IT Security e Software Engineering utilizando um algoritmo de modelagem de tópico chamado Latent Dirichlet Allocation (LDA).

Para conduzir este estudo, foi feito o levantamento de três objetivos, eles são: Compreender como os usuários de diferentes sites de perguntas e respostas, os fóruns, categorizam as questões sobre segurança de software, para isto foi desenvolvido os tópicos e foi identificado as *tags* mais comuns para obter compreensão sobre o que a comunidade está pesquisando. O próximo objetivo foi identificar as perguntas mais difíceis, ou seja, qual o tempo que leva uma pergunta dos fóruns serem respondidas e por fim, o último objetivo que foi o de comparar nossa pesquisa com artigos sobre o mesmo tema.

Com os objetivos citados, foi realizado uma pesquisa sobre como iria ser feito esse levantamento, pelo fato dos fóruns terem um grande número de dados, impossibilitando de ser feito manualmente. Dessa forma, foram encontrados artigos que citavam o LDA, o que acabou se tornando a escolha por ser um algoritmo que lida com uma extensa variedade de registros, algo que estava sendo necessário para continuar com o estudo.

Por ser um algoritmo de aprendizado de máquina, foram definidos os parâmetros do LDA para que os resultados obtidos possam ser reproduzidos posteriormente. Com este algoritmo foi possível alcançar os objetivos, encontrando resultados similares a estudos anteriores. Foram descobertos tópicos como 'Segurança em Aplicações PHP', em que foi possível observar seu crescimento ao decorrer dos anos e também sua queda, resultado que mostra que é um assunto que os desenvolvedores tem tido bastante dificuldade com pico em 2015.

A partir desse estudo, foi possível perceber que o estudo na área de 'Segurança de Software', apesar de ter iniciado em anos anteriores, começou a atrair atenção da comunidade a partir de 2011, e obteve uma crescente na maioria das categorias de perguntas até o ano de 2017, ano em que a quantidade de perguntas começou a reduzir, tendendo a um resultado similar aos anos anteriores a 2016. Além disso, também foi possível perceber que, comparando este estudo com os artigos utilizados como apoio nesta pesquisa, que as *tags* utilizadas pelos usuários seguem bem similares, aparecendo em nosso estudo algumas *tags* que antes não eram tão utilizadas, como *spring boot*, o que era de se esperar devido

à evolução das tecnologias e mudanças no cenário de Engenharia de Software desde 2016, ano em que foi realizado o estudo de Rahman (2016).

Para pesquisas futuras, pode-se utilizar outros algoritmos de aprendizado de máquina para comparar seus resultados, e verificar se obtêm-se os mesmos desdobramentos. Outro acréscimo à pesquisa, poderia adicionar juntamente com a coluna "Title"utilizada, a coluna "Body". Essa coluna é uma explicação do título da pergunta, permitindo obter uma descrição mais clara do que trata a categoria da pergunta em questão.

Referências

ALFAYEZ, R. et al. What is asked about technical debt (TD) on Stack Exchange question-and-answer (Q&A) websites? An observational study. *Empirical Software Engineering*, v. 28, n. 2, p. 35, mar. 2023. ISSN 1382-3256, 1573-7616. Disponível em: <https://link.springer.com/10.1007/s10664-022-10269-5>. Citado 4 vezes nas páginas 17, 41, 42 e 44.

AMARAL, J. J. F. COMO FAZER UMA PESQUISA BIBLIOGRÁFICA. jan. 2007. Citado na página 39.

AQUASEC. *What Is the Secure Software Development Lifecycle (SSDLC)?* 2021. Disponível em: <https://www.aquasec.com/cloud-native-academy/supply-chain-security/secure-software-development-lifecycle-ssdlc/>. Citado na página 26.

BEYER, S.; PINZGER, M. Synonym Suggestion for Tags on Stack Overflow. In: *2015 IEEE 23rd International Conference on Program Comprehension*. [S.l.: s.n.], 2015. p. 94–103. ISSN: 1092-8138. Citado na página 36.

BHUIYAN, T. et al. API vulnerabilities: Current status and dependencies. *International Journal of Engineering and Technology(UAE)*, v. 7, p. 9–13, mar. 2018. Citado na página 23.

BLEI, D. M.; NG, A. Y.; JORDAN, M. I. Latent Dirichlet Allocation. 2003. Citado na página 43.

CARR, S.; MAYO, J.; SHENE, C.-K. Race conditions: a case study. v. 17, p. 90–105, 2001. Citado na página 23.

CZECHOWSKI, A. et al. *Microsoft Security Development Lifecycle*. 2023. Disponível em: <https://learn.microsoft.com/en-us/windows/security/threat-protection/msft-security-dev-lifecycle>. Citado na página 32.

DEEPA, G.; THILAGAM, P. S. Securing web applications from injection and logic vulnerabilities: Approaches and challenges. *Information and Software Technology*, v. 74, p. 160–180, jun. 2016. ISSN 0950-5849. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0950584916300234>. Citado na página 22.

ERDőDI, L.; JOSANG, A. Exploitation vs. Prevention: The Ongoing Saga of Software Vulnerabilities. *Acta Polytechnica Hungarica*, v. 17, p. 199–218, jan. 2020. Citado na página 25.

FALEIROS, T. D. P. *Propagação em grafos bipartidos para extração de tópicos em fluxo de documentos textuais*. Tese (Doutorado em Ciências de Computação e Matemática Computacional) — Universidade de São Paulo, São Carlos, nov. 2016. Disponível em: <http://www.teses.usp.br/teses/disponiveis/55/55134/tde-10112016-105854/>. Citado 2 vezes nas páginas 37 e 38.

HANIF, H. et al. The rise of software vulnerability: Taxonomy of software vulnerabilities detection and machine learning approaches. *Journal of Network and*

Computer Applications, v. 179, p. 103009, abr. 2021. ISSN 1084-8045. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1084804521000369>. Citado 2 vezes nas páginas 16 e 25.

HORA, A. Googling for Software Development: What Developers Search For and What They Find. In: *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*. Madrid, Spain: IEEE, 2021. p. 317–328. ISBN 978-1-72818-710-5. Disponível em: <https://ieeexplore.ieee.org/document/9463121/>. Citado 2 vezes nas páginas 15 e 16.

KHAN, M. U. A.; ZULKERNINE, M. Activity and Artifact Views of a Secure Software Development Process. In: *2009 International Conference on Computational Science and Engineering*. Vancouver, BC, Canada: IEEE, 2009. p. 399–404. ISBN 978-1-4244-5334-4. Disponível em: <http://ieeexplore.ieee.org/document/5283250/>. Citado na página 15.

LIPNER, S.; HOWARD, M. The Security Development Lifecycle. *Datenschutz und Datensicherheit - DuD*, v. 34, n. 3, p. 135–137, mar. 2010. ISSN 1614-0702, 1862-2607. Disponível em: <http://link.springer.com/10.1007/s11623-010-0021-7>. Citado na página 31.

MCGRAW, G. Software Security: Building Security In. In: *2006 17th International Symposium on Software Reliability Engineering*. [S.l.: s.n.], 2006. p. 6–6. ISSN: 2332-6549. Citado na página 15.

MCGRAW, G. Building Security In Maturity Model. 2009. Citado na página 28.

MEUCCI, M. *OWASP SAMM v2: lessons learned after 9 years of assessment*. 2020. Disponível em: <https://blog.mindedsecurity.com/2020/04/owasp-samm-v2-lessons-learned-after-9.html>. Citado na página 30.

MICROSOFT. *Simplified Implementation of the Microsoft SDL*. 2011. Disponível em: <https://www.microsoft.com/en-us/download/details.aspx?id=12379>. Citado na página 32.

MOHAMMED, N. M. et al. Exploring software security approaches in software development lifecycle: A systematic mapping study. *Computer Standards & Interfaces*, v. 50, p. 107–115, fev. 2017. ISSN 0920-5489. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0920548916301155>. Citado na página 15.

NETZER, R. H.; MILLER, B. P. What are race conditions? Some issues and formalizations. *ACM Letters on Programming Languages and Systems (LOPLAS)*, v. 1, n. 1, p. 74–88, 1992. Disponível em: <https://dl.acm.org/doi/abs/10.1145/130616.130623>. Citado na página 23.

OWASP. *SAMM-v2-PDF.pdf*. 2020. Disponível em: <https://drive.google.com/file/d/1cI3Qzfrly_X89z7StLWI5p_Jfqs0-OZv/view?usp=sharing&usp=embed_facebook>. Citado 2 vezes nas páginas 29 e 30.

PIESSENS, F. Software Security. In: *The Cyber Security Body of Knowledge v1.1.0, 2021*. University of Bristol, 2021. KA Version 1.0.1. Disponível em: <https://www.cybok.org/>. Citado 6 vezes nas páginas 20, 21, 22, 23, 24 e 26.

PONZANELLI, L.; BACCHELLI, A.; LANZA, M. Seahawk: Stack Overflow in the IDE. In: *2013 35th International Conference on Software Engineering (ICSE)*. [S.l.: s.n.], 2013. p. 1295–1298. ISSN: 1558-1225. Citado na página 36.

RAHMAN, M. S. *An empirical case study on Stack Overflow to explore developers' security challenges*. [S.l.], 2016. Disponível em: <https://krex.k-state.edu/handle/2097/34563>. Citado 6 vezes nas páginas 17, 41, 46, 64, 65 e 68.

RANSOME, J.; MISRA, A. *Core Software Security: Security at the Source*. [S.l.]: CRC Press, 2018. ISBN 9780429623646. Citado na página 15.

RAPôSO, C. F. L. et al. LGPD - LEI GERAL DE PROTEÇÃO DE DADOS PESSOAIS EM TECNOLOGIA DA INFORMAÇÃO: Revisão Sistemática. *RACE - Revista de Administração do Cesmac*, v. 4, p. 58–67, ago. 2019. ISSN 2675-3766. Disponível em: <https://revistas.cesmac.edu.br/administracao/article/view/1035>. Citado na página 16.

REKHA, V. S.; VENKATAPATHY, S. Understanding the Usage of Online Forums as Learning Platforms. *Procedia Computer Science*, v. 46, p. 499–506, 2015. ISSN 18770509. Disponível em: <https://linkinghub.elsevier.com/retrieve/pii/S1877050915001386>. Citado na página 15.

SADOWSKI, C.; STOLEE, K. T.; ELBAUM, S. How developers search for code: a case study. In: *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. Bergamo Italy: ACM, 2015. p. 191–201. ISBN 978-1-4503-3675-8. Disponível em: <https://dl.acm.org/doi/10.1145/2786805.2786855>. Citado na página 16.

SNYK. *Secure SDLC | Secure Software Development Life Cycle*. 2020. Disponível em: <https://snyk.io/learn/secure-sdlc/>. Citado na página 26.

SYNOPSYS. *What Is the BSIMM and How Does It Work? | Synopsys*. 2019. Disponível em: <https://www.synopsys.com/glossary/what-is-bsimm.html>. Citado na página 27.

SYNOPSYS. *BSIMM13 Foundations*. 2022. Disponível em: <https://www.synopsys.com/software-integrity/engage/bsimm-web/bsimm13-foundations>. Citado na página 27.

Sá-SILVA, J. R.; ALMEIDA, C. D. de; GUINDANI, J. F. Pesquisa documental: pistas teóricas e metodológicas. *Revista Brasileira de História*, 2009. Citado na página 39.

UMAIR, M.; ZULKERNINE, M. On Selecting Appropriate Development Processes and Requirements Engineering Methods for Secure Software. In: . [S.l.: s.n.], 2009. v. 2, p. 353–358. Citado na página 15.

WILLIAMS, L.; MCGRAW, G.; MIGUES, S. Engineering Security Vulnerability Prevention, Detection, and Response. *IEEE Software*, v. 35, n. 5, p. 76–80, set. 2018. ISSN 1937-4194. Citado na página 24.

YANG, X.-L. et al. What Security Questions Do Developers Ask? A Large-Scale Study of Stack Overflow Posts. *Journal of Computer Science and Technology*, v. 31, n. 5, p. 910–924, set. 2016. ISSN 1000-9000, 1860-4749. Disponível em: <http://link.springer.com/10.1007/s11390-016-1672-0>. Citado 2 vezes nas páginas 41 e 44.

ZHOU, Z. et al. *Vulnerable GPU Memory Management: Towards Recovering Raw Data from GPU*. arXiv, 2016. ArXiv:1605.06610 [cs]. Disponível em: <http://arxiv.org/abs/1605.06610>. Citado 2 vezes nas páginas 21 e 22.

Apêndices

APÊNDICE A – Código SQL

Código de consulta de seleção que busca registros na tabela *"Posts"* de cada fórum, com base em determinados critérios de pesquisa.

A consulta busca registros que contenham palavras-chave relacionadas à segurança de software nos campos mencionados. Foi incluido o termo *"Security"* nas *Tags*, pois se fosse adicionado no *Body* ou *Title* poderia resultar em uma série de registros que podem não estar diretamente relacionados à pesquisa de segurança de software.

A cláusula LIKE é usada para realizar comparações parciais, permitindo que registros que contenham qualquer parte dessas palavras-chave sejam incluídos nos resultados. A condição DATALENGTH(Tags) > 0 verifica se o campo 'Tags' possui um comprimento maior que zero, a fim de prosseguir com a etapa de processamento das tags. Essa verificação é realizada para eliminar perguntas que não contenham tags. Por fim, as condições ID IS NOT NULL e ISNUMERIC(ID) $= 1$ referem-se à obrigatoriedade do campo 'ID' estar preenchido e conter exclusivamente valores numéricos. O valor '1' indica que a coluna 'ID' contém valores numéricos.

```
SELECT *
FROM Posts
WHERE (
    UPPER(Tags) LIKE UPPER('%security%')
)
AND DATALENGTH(Tags) > 0
AND ID IS NOT NULL
AND CreationDate < '2023-09-01'
AND ISNUMERIC(ID) = 1;
```

No Stack Overflow (SO), foram executadas três consultas, sendo que em cada uma delas as seguintes linhas foram adicionadas:

```
AND YEAR(CreationDate) > 2016
```

```
AND YEAR(CreationDate) < 2016
```

```
AND YEAR(CreationDate) = 2016
```

Isso se deve ao fato de que o *Stack Exchange Data* impõe um limite de 50 mil linhas por consulta. Nessas últimas duas consultas, foi retirado essa linha do código, pois ela não possui utilidade, e os resultados seriam retornado incorretos.

```
AND CreationDate < '2023-09-01'
```

APÊNDICE B – Gráfico de Categorias por Ano

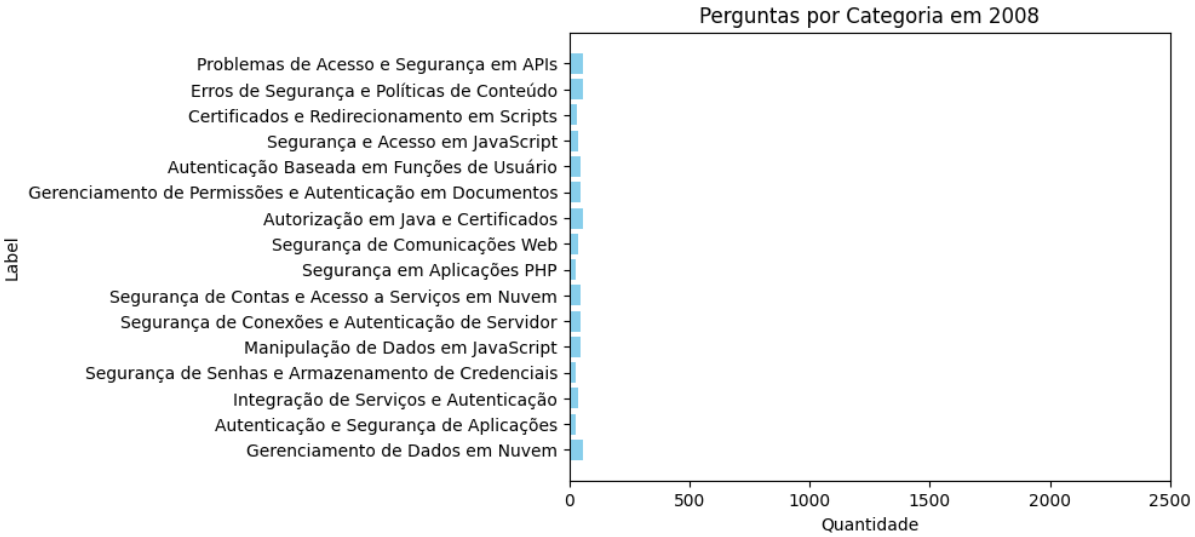

Figura 21 – Quantidade de Perguntas por Categoria em 2008

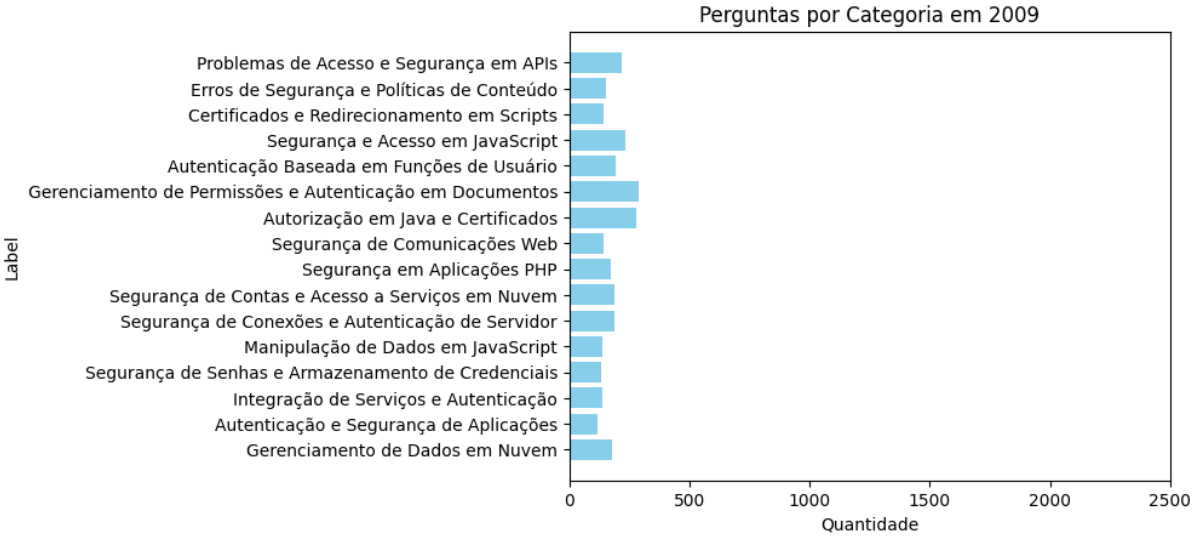

Figura 22 – Quantidade de Perguntas por Categoria em 2009

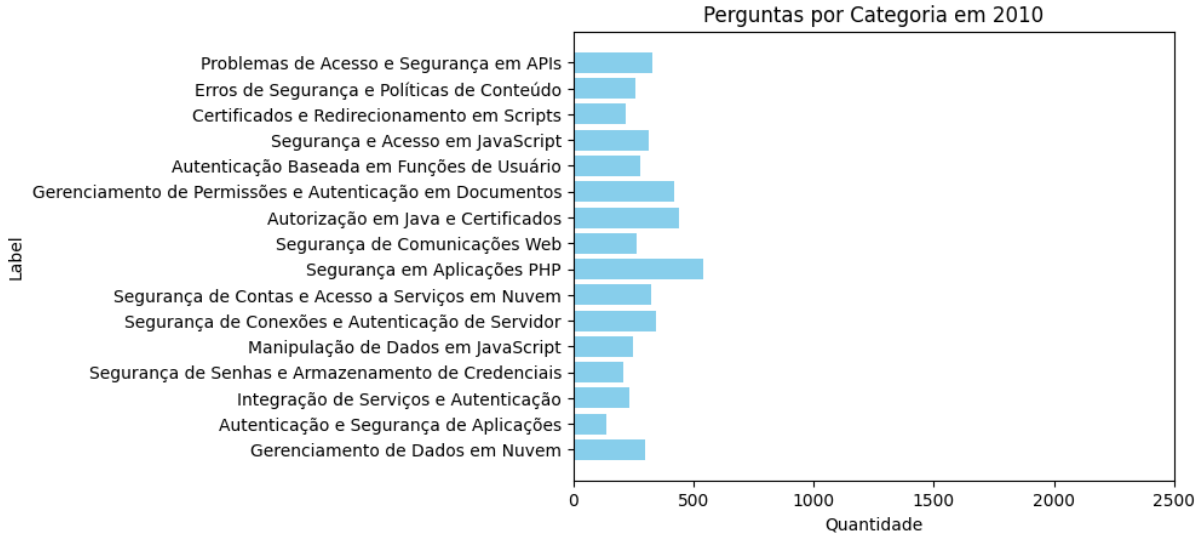

Figura 23 – Quantidade de Perguntas por Categoria em 2010

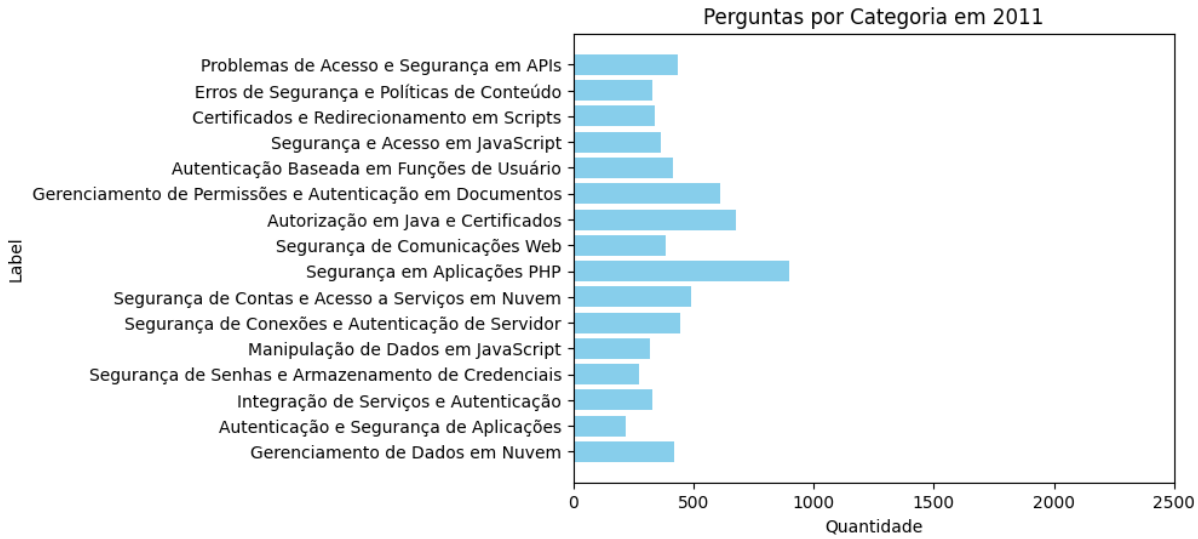

Figura 24 – Quantidade de Perguntas por Categoria em 2011

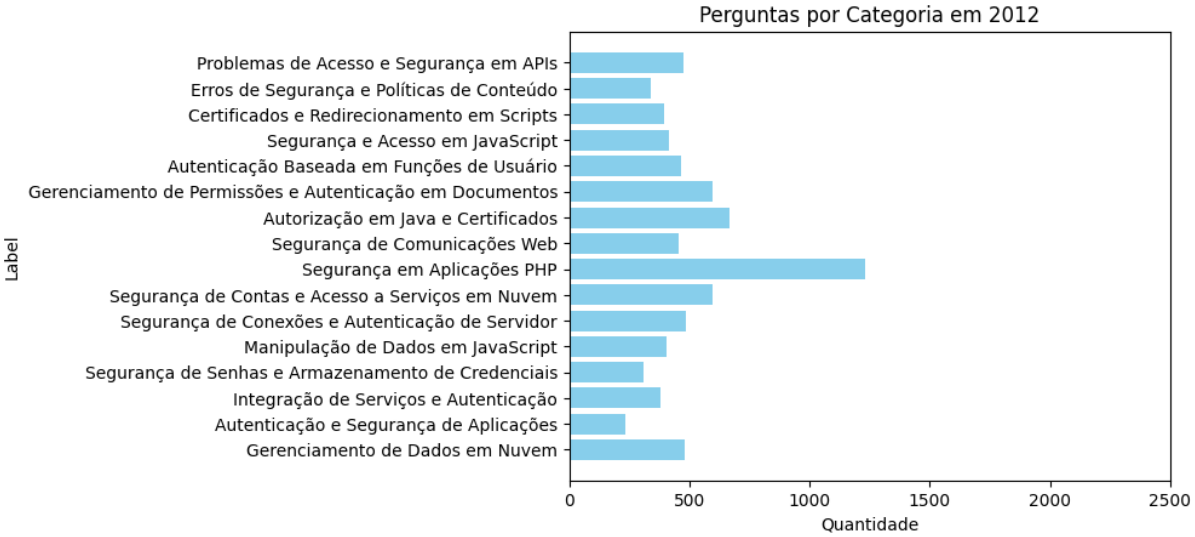

Figura 25 – Quantidade de Perguntas por Categoria em 2012

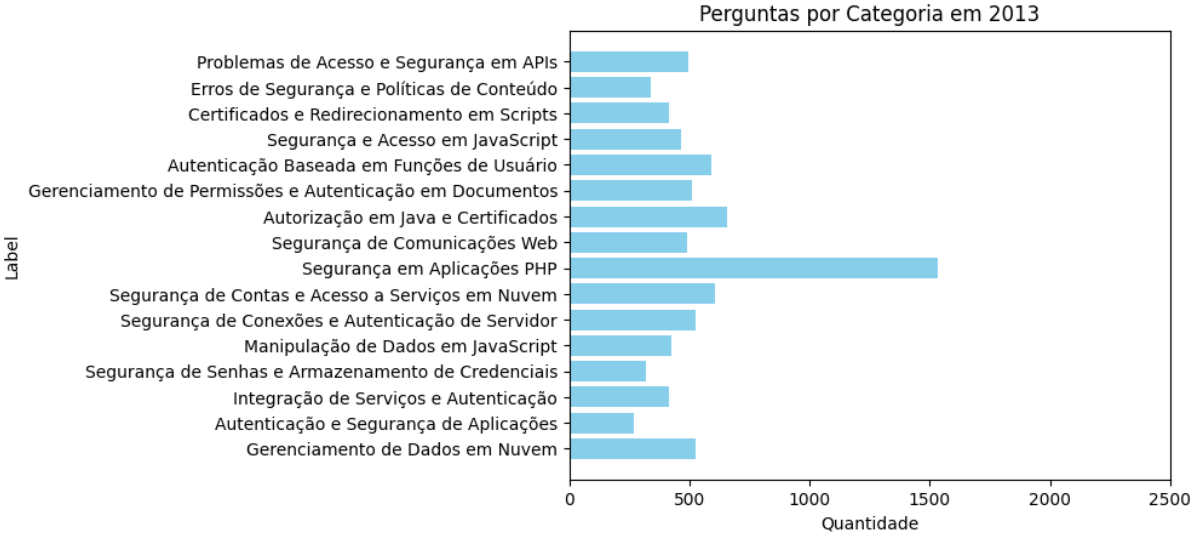

Figura 26 – Quantidade de Perguntas por Categoria em 2013

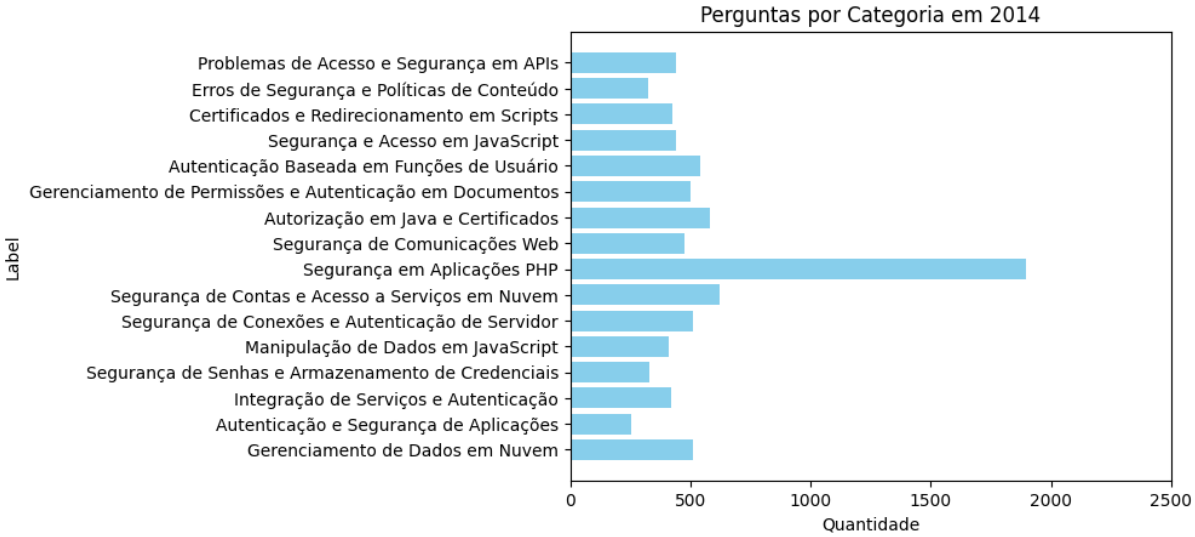

Figura 27 – Quantidade de Perguntas por Categoria em 2014

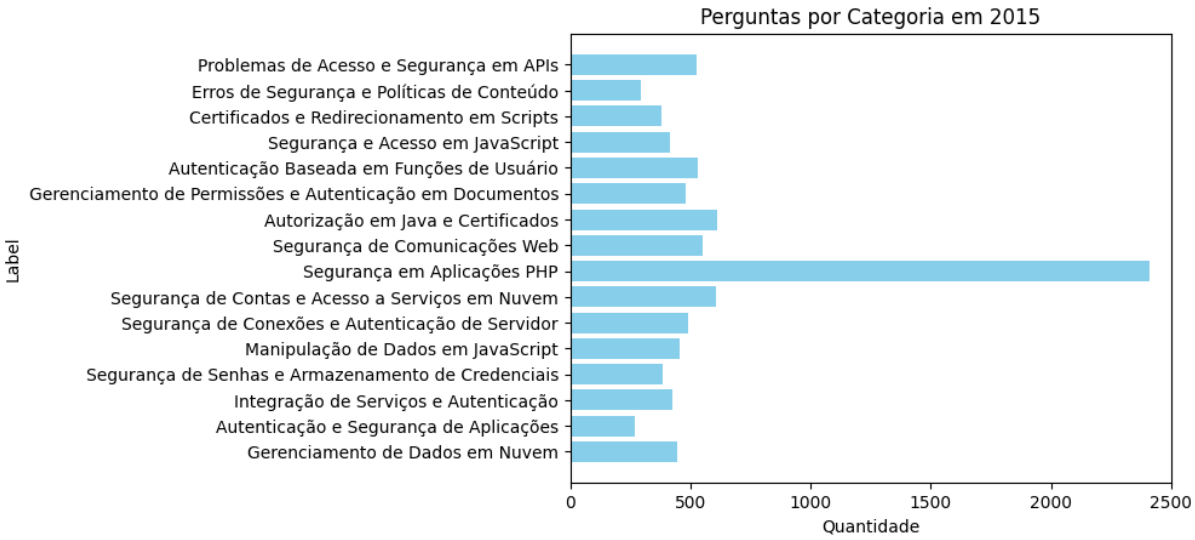

Figura 28 – Quantidade de Perguntas por Categoria em 2015

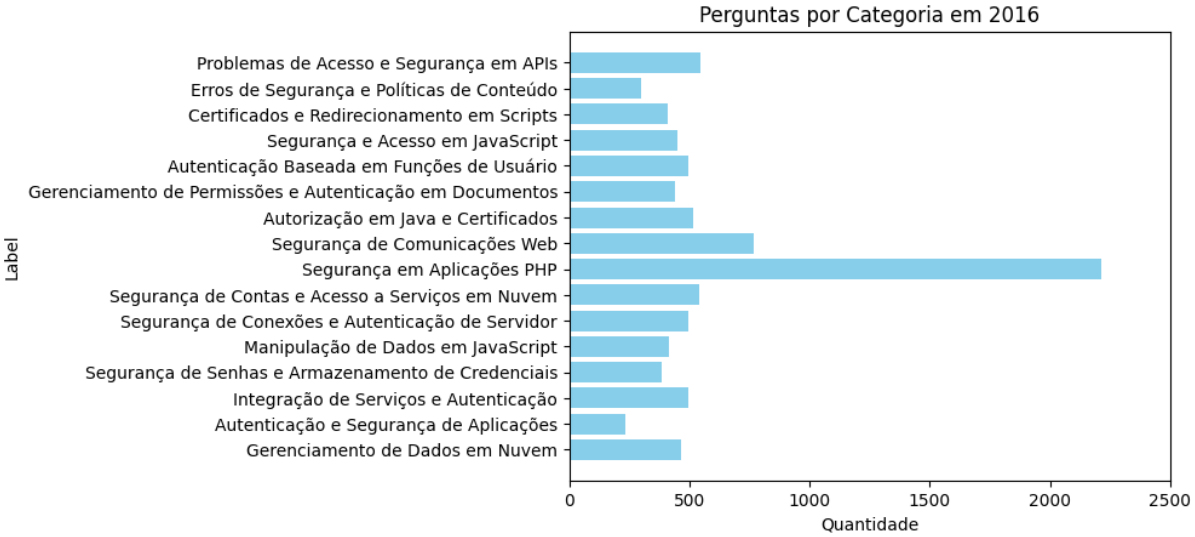

Figura 29 – Quantidade de Perguntas por Categoria em 2016

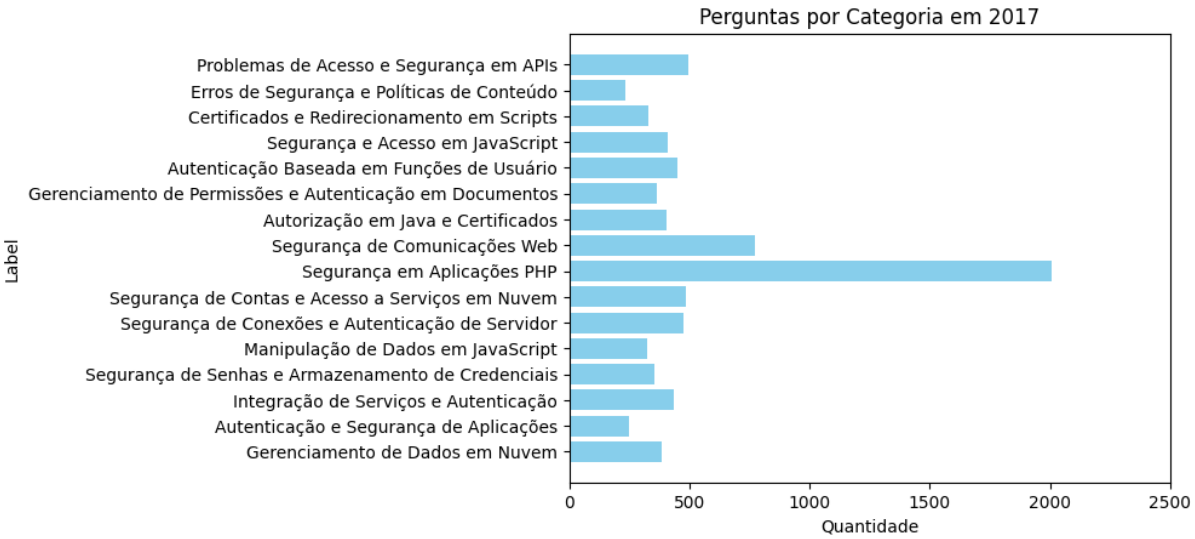

Figura 30 – Quantidade de Perguntas por Categoria em 2017

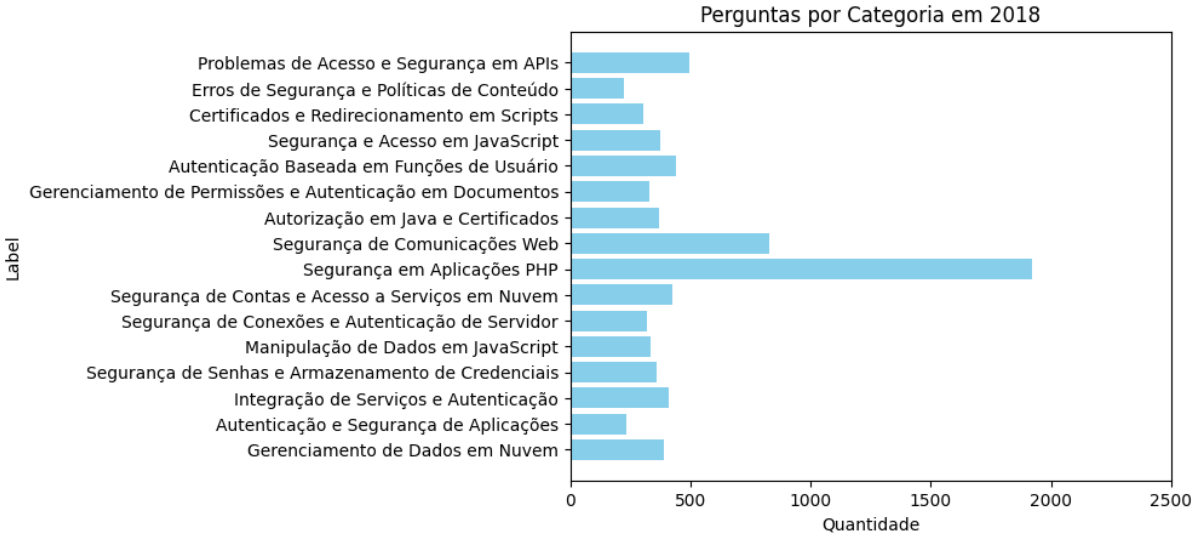

Figura 31 – Quantidade de Perguntas por Categoria em 2018

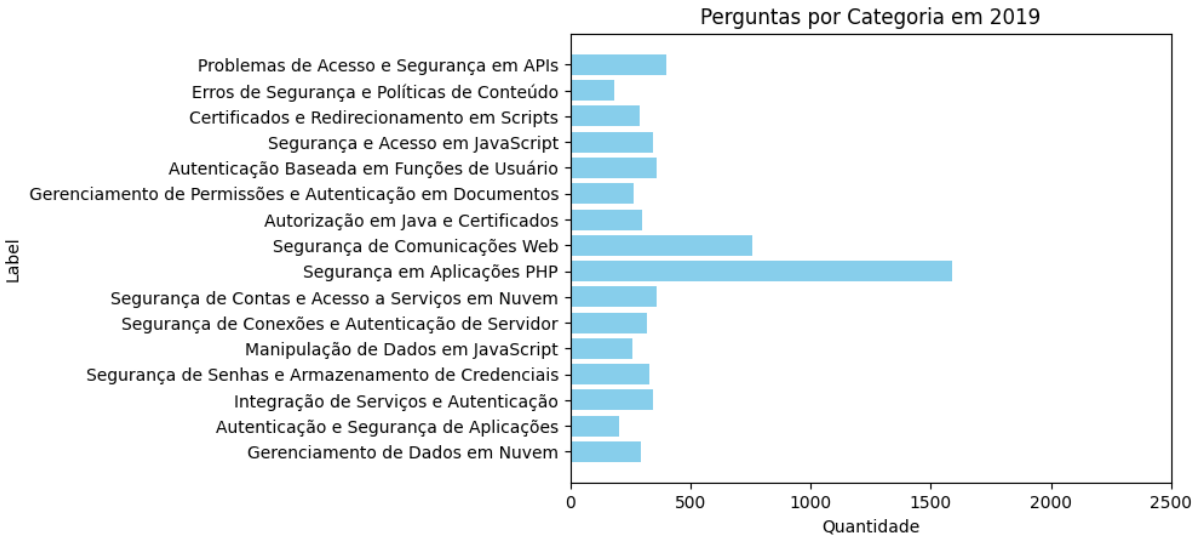

Figura 32 – Quantidade de Perguntas por Categoria em 2019

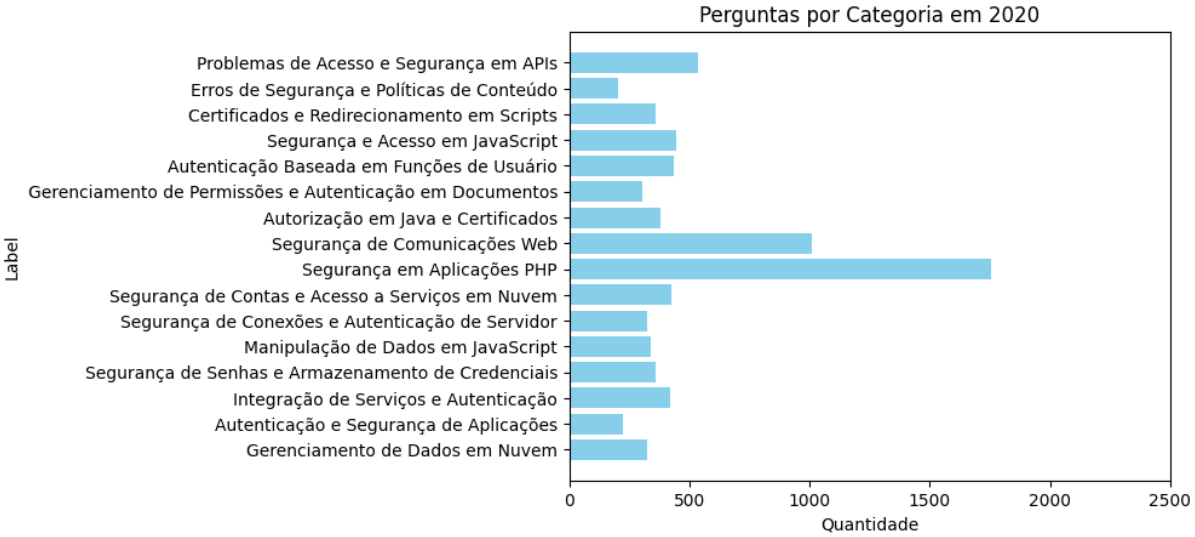

Figura 33 – Quantidade de Perguntas por Categoria em 2020

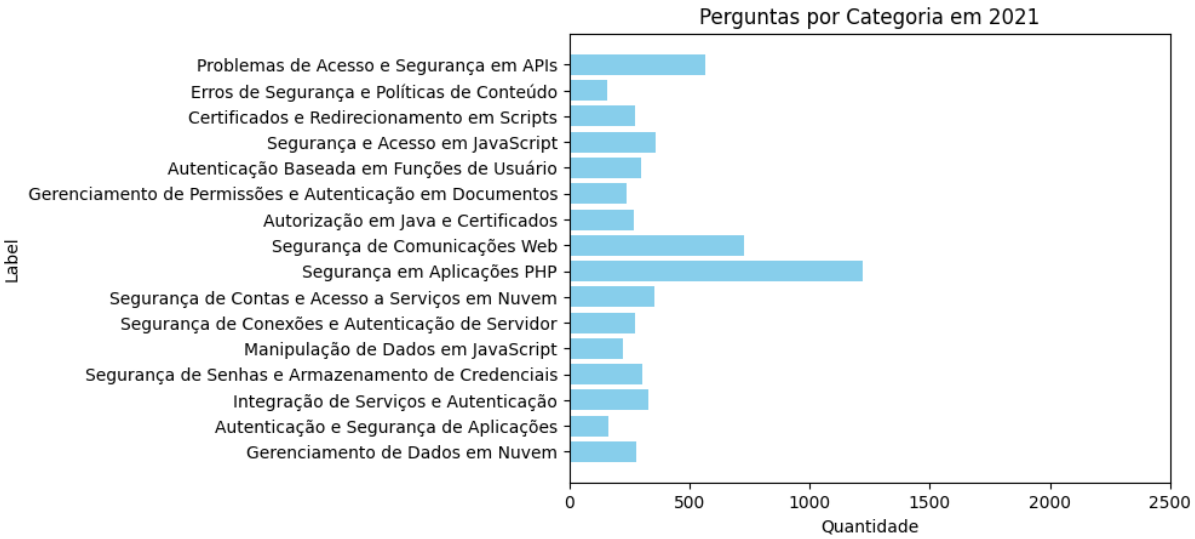

Figura 34 – Quantidade de Perguntas por Categoria em 2021

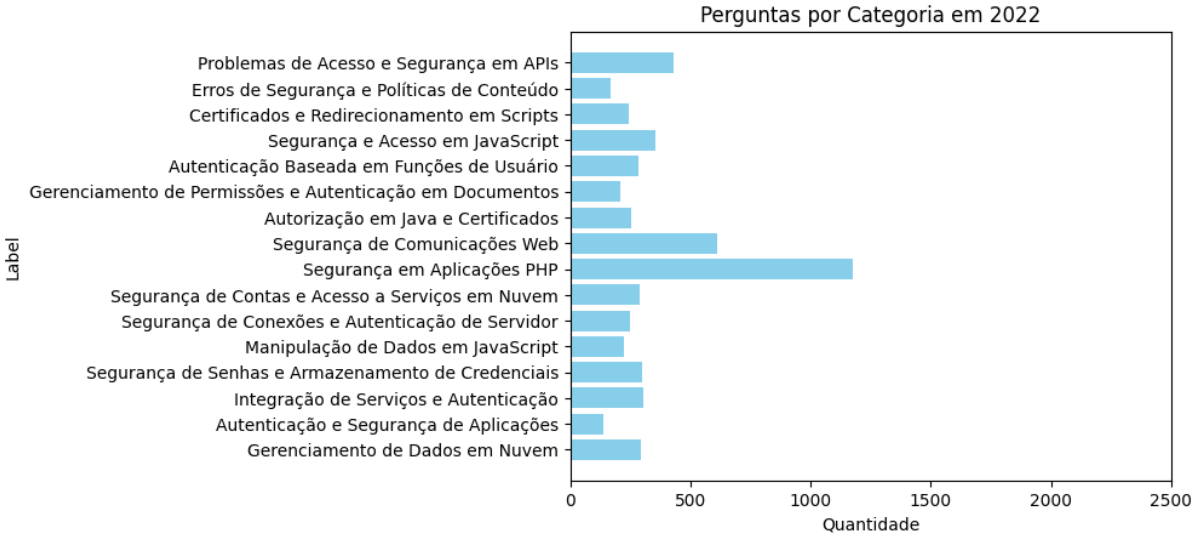

Figura 35 – Quantidade de Perguntas por Categoria em 2022

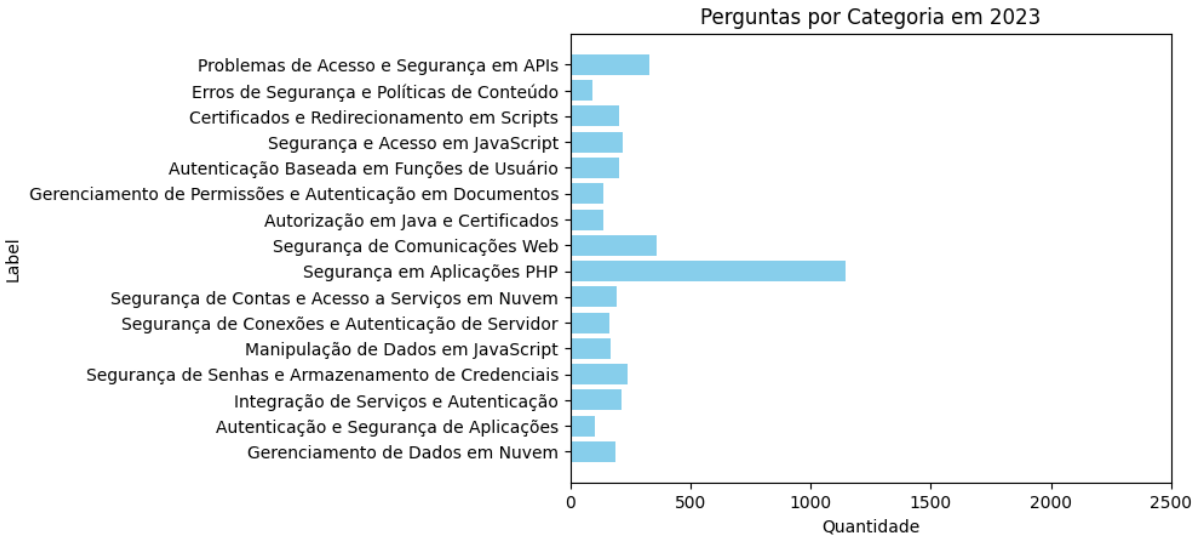

Figura 36 – Quantidade de Perguntas por Categoria em 2023