UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIAS DA COMPUTAÇÃO

AMADOR BUENO ROCHA JUNIOR

Identificação de Vulnerabilidades em Dispositivos Móveis a partir de Sites Especializados e Vídeos na Web usando Web Scraping e LLMs

Recife

2025

AMADOR BUENO ROCHA JUNIOR

Identificação de Vulnerabilidades em Dispositivos Móveis a partir de Sites Especializados e Vídeos na Web usando Web Scraping e LLMs

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação. Área de Concentração: Inteligência Computacional.

Orientador(a): Flávia de Almeida Barros

Recife

2025

.Catalogação de Publicação na Fonte. UFPE - Biblioteca Central

Rocha Junior, Amador Bueno.
Identificação de vulnerabilidades em dispositivos móveis a partir de sites especializados e vídeos na web usando Web scraping e LLMs / Amador Bueno Rocha Junior. - Recife, 2025.
125f.: il.

Dissertação (Mestrado)- Universidade Federal de Pernambuco, Centro de Informática, Programa de Pós Graduação em Ciências da Computação/PPGCC, 2025.
Orientação: Flávia de Almeida Barros.

1. Identificação de vulnerabilidades em smartphones; 2. Processamento de linguagem natural; 3. Mineração de texto; 4. SO Android; 5. Web scraping e LLMs. I. Barros, Flávia de Almeida. II. Título.

UFPE-Biblioteca Central

Amador Bueno Rocha Junior

“Identificação de vulnerabilidades em dispositivos móveis a partir de sites especializados e vídeos na Web usando Web scraping e LLMs”

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação. Área de Concentração: Inteligência Computacional.

Aprovado em: 31/07/2025.

BANCA EXAMINADORA

_____

Prof. Dr. Juliano Manabu Iyoda
Centro de Informática / UFPE

_____

Prof. Dr. Lucas Albertins de Lima
Departamento de Computação / UFRPE

_____

Profa. Dra. Flávia de Almeida Barros
Centro de Informática/UFPE
(**orientadora**)

AGRADECIMENTOS

Em primeiro lugar, agradeço a Deus pela força e proteção ao longo desta caminhada.

À minha mãe, Elisa, por todo amor, apoio, compreensão e incentivo constantes em cada etapa da minha vida. Às minhas filhas, Laura e Luísa, razão maior dos meus esforços e inspiração diária. Ao meu pai, Amador (in memoriam), pelo exemplo e valores deixados, que continuam guiando minhas escolhas.

À minha namorada, Mayara, pelo carinho, companheirismo, compreensão e apoio incondicional, especialmente nos momentos mais desafiadores desta jornada.

Expresso minha gratidão à minha orientadora, professora Flávia Barros, pelos ensinamentos, críticas construtivas, dedicação e paciência, que foram fundamentais para a construção e conclusão deste trabalho, além do meu crescimento acadêmico e pessoal.

Agradeço também aos colegas de mestrado e amigos, pelo incentivo, troca de experiências e parceria ao longo do curso, tornando a trajetória mais leve e enriquecedora.

Ao Centro de Informática da UFPE e ao Projeto CIn-Motorola, pela estrutura e ambiente de aprendizado proporcionados. Em especial, agradeço ao professor Alexandre Mota, Audir Paiva e Rafael Matuo pelo apoio, colaboração e valiosas contribuições durante o desenvolvimento desta pesquisa.

Por fim, registro minha gratidão a todos que, direta ou indiretamente, contribuíram para a realização deste trabalho.

RESUMO

Smartphones modernos armazenam dados sensíveis dos usuários, incluindo informações pessoais, rotas e credenciais bancárias. As ameaças à segurança desses dispositivos são vistas como vulnerabilidades, i.e., fraquezas no hardware ou no software que podem se tornar a porta de entrada para hackers acessarem o dispositivo. Como os testes realizados na fase de pré-venda nem sempre conseguem eliminar todas as vulnerabilidades existentes, empresas buscam também identificar essas vulnerabilidades no pós-venda o mais rápido possível, a fim de evitar que os usuários sejam afetados. Nessa fase, os relatos técnicos disponibilizados na Web por usuários avançados ou hackers são uma rica fonte de informação, pois trazem descrições detalhadas de como quebrar a segurança dos dispositivos, agilizando assim o trabalho dos desenvolvedores. Contudo, considerando a grande quantidade de relatos disponíveis e a alta frequência de novas publicações, o processo manual de busca de informação atualmente adotado em algumas empresas não é eficaz, sendo ainda demorado e dispendioso. Nesse contexto, o objetivo geral deste trabalho foi investigar a identificação automática de vulnerabilidades em sistemas operacionais (SO) de smartphones a partir de relatos e vídeos disponíveis na Internet. Apesar da relevância do tema, não foram identificados, até o momento, trabalhos na literatura com abordagem semelhante focando a extração automatizada de vulnerabilidades via Web. Especificamente, focamos o trabalho no SO Android, que é o mais popular em uso nos smartphones. Um sistema protótipo foi desenvolvido com base em técnicas como Web scraping, Mineração de Textos, Processamento de Linguagem Natural e Modelos de Linguagem de Grande Escala (LLMs) para a coleta e análise de dados relevantes com precisão e rapidez. Os dados tratados são armazenados em um banco de dados, e os resultados são apresentados através de uma interface de consulta. O estudo de caso foi realizado com foco na quebra da Proteção de Redefinição de Fábrica *(Factory Reset Protection* – FRP) para acesso indevido aos smartphones, por se tratar de uma vulnerabilidade crítica. Testes do protótipo com usuários da empresa parceira, Motorola Mobility, demonstraram resultados muito satisfatórios, com uma ótima aceitação por parte dos profissionais que o utilizaram. Este trabalho foi conduzido no contexto de uma colaboração na área de Teste de SW entre o CIn-UFPE e a Motorola Mobility, uma empresa global especializada no desenvolvimento de dispositivos móveis.

Palavras-chaves: Identificação de vulnerabilidades em smartphones. SO Android. Mineração de Texto. Processamento de Linguagem Natural. Web Scraping e LLMs.

ABSTRACT

Modern smartphones store sensitive user data, including personal information, routes, and banking credentials. Security threats to these devices are seen as vulnerabilities — i.e., weaknesses in the hardware or software that can become entry points for hackers to access the device. Since testing conducted in the pre-sale phase cannot always eliminate all existing vulnerabilities, companies also seek to identify these vulnerabilities in the post-sale phase as quickly as possible, in order to prevent users from being affected. At this stage, technical reports made available online by advanced users or hackers are a rich source of information, as they provide detailed descriptions of how to breach device security, thus speeding up the work of developers. However, given the large number of reports available and the high frequency of new publications, the manual information search process currently adopted by some companies is not effective, still being time-consuming and costly. In this context, the general objective of this work was to investigate the automatic identification of vulnerabilities in smartphone operating systems (OS) based on reports and videos available on the Internet. Despite the relevance of the topic, no studies have been identified in the literature to date with a similar approach focusing on automated vulnerability extraction via the Web. Specifically, we focused on the Android OS, which is the most popular in use on smartphones. A prototype system was developed based on techniques such as Web scraping, Text Mining, Natural Language Processing, and Large-Scale Language Models (LLMs) to collect and analyze relevant data accurately and quickly. The processed data are stored in a database, and the results are presented through a query interface. The case study focused on breaching the Factory Reset Protection (FRP) for unauthorized access to smartphones, as this is a critical vulnerability. Testing of the prototype with users of the partner company, Motorola Mobility, demonstrated very satisfactory results, with excellent acceptance by the professionals who used it. This work was conducted as part of a collaboration in the Software Testing area between CIn-UFPE and Motorola Mobility, a global company specializing in the development of mobile devices.

Keywords: Identification of vulnerabilities in smartphones. Android OS. Text Mining. Natural Language Processing. Web Scraping and LLMs.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Os smartphones atuais são, de fato, computadores móveis conectados à Internet. Eles armazenam nossos dados pessoais, trajetos percorridos, aplicativos bancários e outras informações sensíveis. Esse cenário traz ameaças variadas à segurança dos dados e, consequentemente, aos usuários de tais dispositivos. Por um lado, o sistema operacional dos celulares pode ser "invadido" via internet através de práticas como uso de código malicioso (*malware*). Além disso, por se tratarem de dispositivos móveis, esses aparelhos são mais suscetíveis a perdas e furtos. Nesses casos, pessoas mal-intencionadas podem conseguir quebrar a segurança do bloqueio de tela e acessar dados e aplicativos instalados no aparelho.

As ameaças à segurança desses dispositivos são vistas como "vulnerabilidades". De acordo com (KELLEY, 2023), vulnerabilidades de segurança referem-se a "fraquezas" no hardware, no software ou em quaisquer procedimentos, tornando-se portas de entrada para hackers acessarem o sistema.

Infelizmente, os extensivos testes de software realizados nem sempre conseguem garantir a inviolabilidade desses dispositivos, identificando e eliminando todas as vulnerabilidades de segurança existentes. Assim, as empresas buscam alternativas para identificar essas vulnerabilidades no pós-venda, o mais rapidamente possível, a fim de evitar que os usuários sejam afetados. Esses testes são conhecidos como "testes pós-lançamento" (do inglês, *post-releasing tests*)[1]. Nessa fase, são empregadas estratégias variadas, como testes tipo "caixa preta" e análises baseadas em relatos de usuários.[2]

1.1 MOTIVAÇÃO E CONTEXTO GERAL DO TRABALHO

Com a crescente disponibilização, na Web, de relatos de vulnerabilidades identificadas por usuários ou hackers, uma nova estratégia de detecção ganhou importância. Utilizando mecanismos de busca e outros meios disponíveis, colaboradores buscam sites com informações relevantes, a fim de identificar vulnerabilidades ainda desconhecidas pela empresa. Exemplos de sites relevantes incluem: FRP Bypass Files[3], fóruns específicos (e.g., Stack Overflow[4]), canais no YouTube e mídias sociais (como Facebook, X, TikTok). Esses relatos trazem descrições

[1] https://istqb-glossary.page/post-release-testing/
[2] https://www.softwaretestinghelp.com/post-release-testing/
[3] https://bypassfrpfiles.com/
[4] https://stackoverflow.com/questions

detalhadas de como quebrar a segurança dos dispositivos em questão, agilizando assim o trabalho dos colaboradores que precisam eliminar problemas que comprometem a confiabilidade desses aparelhos.

Considerando a grande quantidade de relatos disponíveis e a alta frequência com que esses dados são publicados, fica evidente que o processo manual de busca e extração de informação, atualmente adotado em algumas empresas, não é eficaz, sendo demorado e dispendioso. Esse cenário revela a necessidade de uma solução automatizada para identificação e extração dos dados relevantes, tornando mais eficiente e viável o rápido aproveitamento dos relatos disponíveis.

Contudo, apesar da grande relevância dessa estratégia, a pesquisa bibliográfica realizada até o momento não identificou trabalhos na área que investiguem a automação da identificação e extração de dados relevantes sobre vulnerabilidades em dispositivos móveis a partir de sites na Web.

1.2 CONTEXTO ESPECÍFICO

Este trabalho de mestrado foi conduzido no contexto de uma colaboração entre o Centro de Informática da UFPE (CIn-UFPE) e a Motorola Mobility, uma empresa global especializada no desenvolvimento de dispositivos móveis que utilizam o sistema operacional Android. O foco principal da colaboração entre o CIn-UFPE e a empresa parceira é em teste de software.

Dentre os diversos tipos de testes realizados pela empresa, este trabalho aborda uma lacuna no processo de busca e tratamento de informação extraída de sites na Web sobre vulnerabilidades. Como mencionado, há um interesse crescente na automação da exploração desses relatos na etapa de pós-venda, configurando um tema relevante e atual que ainda não foi estudado com a profundidade necessária.

1.3 TRABALHO REALIZADO

A pesquisa realizada abrangeu estudos teóricos e práticos nas áreas relacionadas ao tema central do trabalho: automação da identificação e extração de informações relevantes sobre vulnerabilidades a partir de relatos na Web. Como resultado, apresentamos este texto e um protótipo, que foi implementado e testado pelas equipes de segurança da empresa parceira.

O estudo de caso realizado com o protótipo desenvolvido teve como foco a identificação

de vulnerabilidades no SO Android, utilizado nos smartphones da Motorola. Vale frisar que a relevância dessa escolha também se justifica pelo fato de que o SO Android é utilizado na maioria dos dispositivos móveis atualmente no mercado[5].

Contudo, como será visto nos capítulos 4 e 5, a solução proposta também pode ser empregada para buscar vulnerabilidades em outros produtos e outros sistemas operacionais, mediante ajustes nos parâmetros de entrada do sistema implementado.

1.3.1 Objetivos do Trabalho e Metodologia de Desenvolvimento

O objetivo geral deste trabalho de mestrado foi realizar um estudo na área de identificação de vulnerabilidades em sistemas operacionais (SO) para dispositivos móveis. Investigamos técnicas para extração automática de dados sobre vulnerabilidades a partir de relatos em sites especializados e em transcrições de vídeos no YouTube. Especificamente, focamos nosso trabalho no Android, que é o sistema operacional mais popular em uso em smartphones.

As técnicas investigadas e utilizadas são oriundas das áreas de Inteligência Artificial (IA) e Mineração de Texto (MT), com foco em Processamento de Linguagem Natural (PLN) e Extração de Informação com Web scraping, visando à extração automática de dados relevantes a partir de sites na Web.

Os objetivos específicos do trabalho são derivados das Questões de Pesquisa (QP) que guiaram a pesquisa realizada, conforme descrito a seguir. Essas QP serão revisitadas na Seção 6.1 do Capítulo 6, que conclui este documento.

- **Questão de Pesquisa 1:** Quais técnicas de IA, PLN e Web scraping são úteis e eficazes na extração e no processamento de dados relevantes sobre vulnerabilidades descritas em formato textual? Essa questão de pesquisa trata da obtenção dos dados de interesse.
- **Questão de Pesquisa 2:** Como organizar, armazenar, atualizar e apresentar os dados extraídos aos usuários finais? Essa questão de pesquisa trata da utilização inteligente e organizada dos dados extraídos.

Para alcançar os objetivos delineados a partir das Questões de Pesquisa acima, realizamos pesquisa teórica e estudos empíricos, através da implementação de um protótipo.

[5] https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/

A metodologia de desenvolvimento do trabalho seguiu os passos descritos a seguir, os quais também resumem os resultados obtidos em cada etapa, compondo as principais contribuições deste trabalho.

1. Estudo de métodos e técnicas das áreas correlatas, com foco em Mineração de Texto com PLN e Web scraping (Questão de Pesquisa 1). Também foi importante estudar métodos de armazenamento eficaz e apresentação dos dados (Questão de Pesquisa 2). Esses estudos foram realizados por meio de cursos, leituras de material didático e de publicações de outros autores que também investigam identificação de vulnerabilidades em SO para dispositivos móveis. Como resultados, foi possível consolidar o conhecimento sobre as técnicas mais relevantes para o nosso caso, incluindo a nova tendência de uso dos Modelos de Linguagem (do inglês, Large Language Models – LLM) para processamento de texto.

2. Pesquisa bibliográfica para busca de trabalhos relacionados na área de Teste de Software, que é a atividade responsável por identificar vulnerabilidades, defeitos e falhas em produtos de SW. Essa pesquisa bibliográfica não localizou publicações de trabalhos baseados em análise de texto para identificação de vulnerabilidades, o que indica a originalidade do tema escolhido.

3. Desenvolvimento de um protótipo para automatizar a extração de dados de vulnerabilidades (relacionados à Questão de Pesquisa 2). A linguagem Python foi escolhida para essa implementação por ser adequada ao tratamento automático de textos e por oferecer uma ampla variedade de bibliotecas para Mineração e Extração de informação textual com PLN. Utilizamos também bancos de dados para armazenar, de forma organizada, os dados extraídos. Por fim, a interface foi desenvolvida usando o framework Flask.

4. Escolha dos sites e canais do YouTube a serem utilizados como fonte de textos para a realização de um estudo de caso, etapa que também contribuiu para responder à Questão de Pesquisa 2, já que o sistema precisava ser adaptável a diferentes formatos de entrada. Inicialmente, focamos no site FRP Bypass Files, que trata de uma vulnerabilidade crítica relacionada à quebra da proteção de fábrica para acesso aos smartphones. A seguir, passamos a explorar vídeos de canais específicos do YouTube, que também trata da mesma vulnerabilidade. Vale salientar que essas escolhas ocorreram simultaneamente ao desenvolvimento do protótipo, já que o código do Web scraping precisa ser ajustado ao

formato dos dados de entrada. Contudo, como será visto nos capítulos 4 e 5, a solução desenvolvida pode ser facilmente adaptada a outros sites Web.

5. Testes quantitativo e qualitativo do protótipo desenvolvido (também vinculados à Questão de Pesquisa 2) revelaram uma taxa de precisão satisfatória no que concerne o cálculo de similaridade entre pares de artigos extraídos da Web. O objetivo aqui é auxiliar os usuários a identificar rapidamente artigos diferentes que tratam da mesma vulnerabilidade. O sistema protótipo já está em uso na empresa parceira (Motorola Mobility), tendo obtido uma alta taxa de acerto nas extrações de dados, bem como uma ótima aceitação por parte dos colaboradores que o utilizam.

1.4 ORGANIZAÇÃO DO DOCUMENTO

O presente texto de dissertação está dividido em mais 5 capítulos, descritos a seguir.

- Capítulo 2 – Apresenta conceitos fundamentais sobre áreas relacionadas ao tema deste trabalho, com foco em Teste de software (com foco principal na identificação e nos tipos de vulnerabilidades existentes) e em Mineração de Texto (relacionada aos processos de aquisição de textos, com destaque ao Web scraping, extração e processamento dos dados relevantes utilizando técnicas de PLN e mineração de padrões).

- Capítulo 3 – Concentra-se no tema de identificação e tratamento de vulnerabilidades de sistemas operacionais para dispositivos móveis. Serão apresentados brevemente o iOS e o Android, bem como suas vulnerabilidades. Serão apresentados os métodos utilizados na identificação de vulnerabilidades, com destaque para a exploração de relatos na Web como sendo um novo caminho a investigar. Por fim, o capítulo descreve brevemente alguns trabalhos relacionados ao tema geral da nossa pesquisa de mestrado, com foco na identificação de vulnerabilidades.

- Capítulo 4 – Apresenta, inicialmente, o contexto do trabalho realizado, seguindo com uma visão geral do processo geral proposto, cujo objetivo é identificar vulnerabilidades a partir da análise de relatos publicados na Web. O processo geral conta com 5 módulos distintos: coleta de dados, pré-processamento dos textos, mineração de dados, armazenamento, e visualização e análise dos resultados. Serão fornecidos detalhes de cada

módulo, bem como o fluxo de dados, o armazenamento e a apresentação dos resultados através da interface do usuário.

- Capítulo 5 – Apresenta o protótipo implementado a partir do processo proposto. Cada módulo será apresentado com detalhes de implementação, trazendo ainda exemplos de uso da interface do usuário. O sistema foi testado por usuários da empresa parceira, tendo obtidos ótimos resultados.

- Capítulo 6 – Apresenta as conclusões e indicações para trabalhos futuros que poderão ser derivados do que foi apresentado aqui.

2 CAPÍTULO 2: ÁREAS DE ESTUDO RELACIONADAS

Este capítulo aborda áreas de estudo relacionadas ao trabalho de pesquisa realizado. Inicialmente, a Seção 2.1 apresenta brevemente conceitos básicos da área de Teste de Software, que é parte essencial no desenvolvimento de qualquer sistema implementado. Destacamos aqui a Seção 2.1.3, que foca em conceitos sobre vulnerabilidades dos softwares, com a apresentação dos principais tipos e técnicas usadas para sua detecção. O Capítulo 3 irá aprofundar esse tema com trabalhos relacionados à área de identificação de vulnerabilidades em Sistemas Operacionais para dispositivos móveis.

A Seção 2.2 introduz conceitos sobre a grande área de Mineração de Texto, com destaque para três tarefas: coleta dos textos (Seção 2.2.1), em particular, Seção 2.2.1.1 sobre *Web Scraping*; o pré-processamento de textos (Seção 2.2.2); e a Seção 2.2.4, que trata da mineração de padrões relevantes nos textos coletados. Como será detalhado no Capítulo 4, essas são as principais tarefas de processamento automático de textos realizadas na solução aqui proposta para identificação de vulnerabilidades.

Além disso, há outros temas relevantes para o contexto deste trabalho, como as tecnologias utilizadas para armazenamento dos dados coletados e tratados, via uso de um banco de dados, e o *framework* usado para desenvolver a interface Web de consulta e atualização dos dados. Contudo, essas tecnologias foram apenas usadas no trabalho, não havendo contribuição de pesquisa nessas áreas.

2.1 TESTE DE SOFTWARE – VISÃO GERAL

Os testes de software são uma etapa essencial do Ciclo de Vida do Desenvolvimento de Software (do inglês, *Software Development Life Cycle* - SDLC) e são fundamentais para garantir a qualidade e a segurança de qualquer produto (SOMMERVILLE, 2015; MYERS; SANDLER; BADGETT, 2011). Assim, torna-se essencial compreender o ciclo de desenvolvimento e os tipos de testes utilizados para garantir a segurança e a funcionalidade dos sistemas.

Esta seção apresenta brevemente conceitos básicos da área de desenvolvimento e teste de software, com foco em vulnerabilidades de software, que é o tema central desta dissertação. Não serão apresentadas aqui ferramentas de teste, uma vez que não adotamos essa abordagem na solução proposta nesta dissertação.

2.1.1 Ciclo de Vida do Desenvolvimento de Software

O Ciclo de Vida de Desenvolvimento de Software (*SDLC*) inclui várias etapas críticas para garantir que o software seja desenvolvido de maneira eficiente e segura. As etapas típicas do SDLC incluem (SOMMERVILLE, 2015; GEEKSFORGEEKS, 2025):

- **Planejamento e Análise de Requisitos:** Coleta das demandas dos clientes e pesquisa de mercado.
- **Definição de Requisitos:** Detalhamento e aprovação dos requisitos com clientes e *stakeholders*; Especificação de todos os requisitos no documento de especificação de requisitos de software (do inglês, *Software Requirements Specification* - SRS).
- **Design da Arquitetura:** Criação do projeto do software com base nos requisitos definidos no SRS, utilizando um Documento de Especificação de Design (do inglês, *Design Document Specification* - DDS).
- **Desenvolvimento do Produto:** Implementação do software conforme o projeto especificado, utilizando linguagens de programação e ferramentas apropriadas.
- **Teste e Integração do Produto:** Teste para garantir a execução correta do software e a correção de falhas.
- **Implantação e Manutenção do Produto:** Lançamento e supervisão do produto após a implantação.

A Figura 1, baseada em (GEEKSFORGEEKS, 2025), apresenta um detalhamento dos estágios do ciclo de desenvolvimento de Software.

Considerando o foco deste trabalho de pesquisa, a seção a seguir irá detalhar apenas conceitos relacionados a Teste de Software.

2.1.2 Testes de Software

Os testes de software podem ser classificados em várias categorias, cada uma com um objetivo específico para garantir a funcionalidade e a segurança do software (HOODA; CHHILLAR, 2015). Esta seção traz alguns conceitos básicos dessa área.

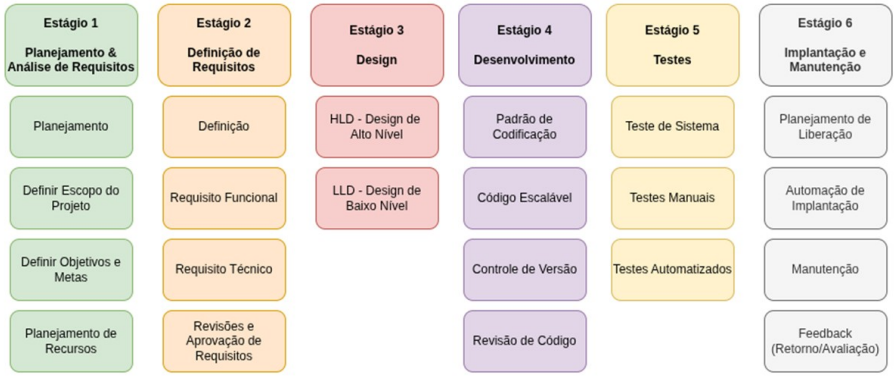

Figura 1 – Estágios do ciclo de desenvolvimento de Software. Fonte: (GEEKSFORGEEKS, 2025).

Inicialmente, podemos distinguir o *modo de execução dos testes* entre execução manual e automática. *Testes manuais* são executados por testadores humanos, sendo úteis para detecção visual rápida de bugs, não exigindo habilidades avançadas de programação por parte dos testadores. Já os *testes automáticos* envolvem a criação de scripts que automatizam o processo de teste, melhorando a eficiência e a cobertura dos testes. A automação de testes permite a execução repetitiva de casos de teste, buscando garantir a consistência e a precisão dos resultados.

As *abordagens de teste* são classificadas entre caixa branca, caixa preta e caixa cinza. Testes de *caixa branca* concentram-se na estrutura interna e na lógica do código. Envolvem a execução do software e a validação de seu comportamento com acesso ao código fonte. Já a abordagem de *caixa preta* foca em avaliar o funcionamento do software sem observar a estrutura interna do código. Por fim, teste de *caixa cinza* combina elementos de teste de caixa branca e caixa preta, oferecendo uma abordagem mais abrangente com uma maior cobertura de código. As três abordagens de teste podem ser realizadas de modo manual ou automático, sendo uma escolha da equipe de testes.

Podemos também distinguir os testes entre funcionais e não funcionais. *Testes funcionais* avaliam funções ou recursos específicos do software, incluindo aqui teste de unidade, teste de integração, teste de sistema, entre outros. Já os *testes não funcionais* avaliam o desempenho, a usabilidade, a confiabilidade e a segurança do software. Destacamos aqui os *testes de segurança*, que buscam identificar vulnerabilidades dos softwares.

Por fim, observamos outra classificação dos testes, entre estáticos, dinâmicos e híbridos.

Testes estáticos seguem a abordagem caixa branca, realizando análise de artefatos de software (como código ou documentos) sem executá-los. Exemplos incluem revisão de código e análise estática de código. Já os *testes dinâmicos* focam nos testes de unidade, de integração e de sistema, podendo ser caixa preta ou caixa branca. Por fim, *testes híbridos* (caixa cinza) combinam testes de caixa preta e caixa branca. Os testadores têm algum conhecimento da estrutura interna, mas também focam na funcionalidade. Exemplos de testes híbridos são *testes de penetração*, que utilizam conhecimentos internos para *identificar vulnerabilidades* enquanto validam a funcionalidade externa.

2.1.3 Vulnerabilidades de Software

Vulnerabilidades de software são *falhas* ou *defeitos* que permitem ataques via comandos maliciosos, acesso a dados sem autorização, ou interrupção de operações normais. A identificação e mitigação dessas vulnerabilidades é essencial, especialmente considerando-se o aumento no uso de dispositivos móveis e a consequente ampliação das possibilidades de ataque (ENCK et al., 2011; SHABTAI et al., 2012).

Como visto acima, vulnerabilidades de software são vistas como *falhas* que podem ser exploradas para comprometer a segurança de um sistema, diferindo de *erros* e *bugs* de código. *Erros* são vistos como problemas que causam o funcionamento incorreto de uma funcionalidade do software (por exemplo, uma falha ao enviar uma foto via WhatsApp). Já os *bugs* são defeitos no software que causam comportamentos indesejados, são problemas lógicos que afetam o desempenho ou a usabilidade do software, mas não necessariamente a sua segurança.

Vulnerabilidades variam desde injeção de SQL até falhas de execução remota de código. A criticidade é frequentemente determinada pelo potencial de dano que uma exploração bem-sucedida pode causar, influenciando diretamente as prioridades de segurança das organizações (ZHOU; JIANG, 2012; FARUKI et al., 2015). Veremos a seguir os tipos mais comuns de vulnerabilidades.

- **Injeção de SQL:** Inserção de código SQL malicioso em entradas de consulta a bancos de dados. Esse ataque envolve a manipulação de entradas do usuário para executar consultas SQL não autorizadas, permitindo que um invasor acesse, modifique ou exclua dados confidenciais armazenados em um banco de dados.

- **Estouro de Buffer:** Ocorre quando um programa tenta gravar mais dados em um buffer

do que este pode conter, resultando no excesso de dados sendo gravados em locais de memória adjacentes. Isso pode permitir que um invasor sobrescreva dados críticos ou injete código malicioso.

- **Cross-Site Scripting (XSS):** Inserção de scripts maliciosos em páginas web vistas por outros usuários.
- **Cross-Site Request Forgery (CSRF):** Permite que um atacante induza o usuário a executar ações indesejadas em um aplicativo Web no qual está autenticado, como alterar configurações ou roubar dados.
- **Insecure Direct Object References (IDOR):** Vulnerabilidades IDOR ocorrem quando um aplicativo expõe referências de objetos internos sem verificações de autorização adequadas. Os invasores podem explorar essas vulnerabilidades para obter acesso não autorizado a dados confidenciais ou executar ações que deveriam ser restritas.
- **Authentication Bypass:** Permite que invasores evitem os mecanismos de autenticação de um sistema de software. Pode resultar em acesso não autorizado, exposição de dados ou aumento de privilégios. Como exemplo importante desse caso, temos o FRP Bypass:
  - **FRP Bypass:** é uma técnica ou conjunto de técnicas utilizadas para contornar a Proteção de Redefinição de Fábrica em dispositivos Android, permitindo que os dados do usuário original sejam redefinidos indevidamente em caso de perda ou furto. Essa vulnerabilidade será melhor detalhada na Seção 2.1.3.1 e no Capítulo 4 deste documento.

Diversas técnicas têm sido empregadas buscando identificar as vulnerabilidades que podem ocorrer em um software complexo, variando entre análise estática, dinâmica e híbrida (Seção 2.1.2). Como já mencionado, a *análise estática* (caixa branca) analisa o código fonte sem execução. Aqui, o objetivo é a identificação precoce de fraquezas estruturais do software. Ferramentas de análise estática detectam padrões conhecidos de vulnerabilidades, como *buffer overflow* e *injeção de código*.

Já a *análise dinâmica de código* executa o código em um ambiente controlado para capturar comportamentos em tempo de execução. O objetivo é detectar vulnerabilidades que só aparecem durante a execução, como problemas de validação de entrada.

Por fim, os *testes de penetração* simulam ataques reais para identificar fraquezas exploráveis. Esses testes fornecem *insights* práticos sobre a segurança do software e orientam esforços para mitigação das vulnerabilidades identificadas.

2.1.3.1 Factory Reset Protection (FRP)

Apesar de todos os esforços para proteger os dispositivos Android, algumas vulnerabilidades ainda podem ser identificadas, permitindo que atacantes (hackers) quebrem sua proteção. Nesse contexto, a Proteção de Redefinição de Fábrica (*Factory Reset Protection* - FRP) foi introduzida pela Google como uma camada adicional de segurança no sistema operacional Android, a partir da sua versão 5.1[1]. Seu principal objetivo é proteger os dispositivos em caso de roubo ou perda, exigindo que o usuário faça login com a conta Google previamente registrada após uma redefinição de fábrica. Isso diminui a atratividade de celulares Android para ladrões, além de proteger os dados pessoais dos usuários mesmo em caso de perda.

A FRP exige que as credenciais do proprietário anterior do dispositivo sejam inseridas durante o processo de configuração se os dispositivos foram redefinidos usando o modo de recuperação. Isso é feito usando uma partição separada, normalmente `/dev/block/.../by-name/frp`, que armazena os dados necessários em um bloco de dados persistente (PDB). Quando um PDB está presente durante o processo de configuração do dispositivo, as credenciais do usuário anterior são verificadas. Se a verificação falhar, o dispositivo não inicializará. Usuários legítimos podem limpar o PDB inicializando uma redefinição de fábrica na interface de configurações do dispositivo ou removendo todas as contas da Google do dispositivo. Um novo usuário não pode fazer isso, mesmo que tenha acesso ao dispositivo desbloqueado, pois ambas as ações exigem as credenciais do usuário anterior.

O Android protege aplicativos e seus dados usando recursos existentes do kernel Linux[2], entre outros. Cada aplicativo é executado em um processo separado e recebe um ID de usuário exclusivo, o que impede que os aplicativos acessem recursos para os quais não têm permissão. Desde o Android 5, o SELinux (*Security Enhanced Linux*[3]) é usado para limitar ainda mais o acesso a outros processos e arquivos. Por padrão, usuários e aplicativos não têm permissões de *root* no Android (LANG, 2022).

Depois do lançamento do FRP, começaram a surgir ataques de *FRP Bypass* buscando

[1] https://www.androidcentral.com/factory-reset-protection-what-you-need-know
[2] https://www.kernel.org/
[3] https://source.android.com/docs/security/features/selinux?hl=pt-br

quebrar a proteção extra oferecida. O objetivo do *FRP Bypass* é desativar a proteção que exige que o usuário faça login com a conta Google previamente registrada, permitindo o acesso ao dispositivo sem as credenciais do usuário anterior. As técnicas de FPR Bypass podem envolver o uso de ferramentas de software, exploração de vulnerabilidades no sistema operacional, manipulação de configurações de acessibilidade, entre outros métodos[4].

A fim de evitar o FRP Bypass, o trabalho de (LEE et al., 2021) propôs uma nova abordagem baseada em um serviço de desbloqueio remoto. A ideia é evitar interferir nos recursos de segurança existentes, preservando assim o nível de segurança atual. Dessa forma, o acesso pode ser recuperado em situações em que a senha foi esquecida, por exemplo.

2.2 MINERAÇÃO DE TEXTO

A Mineração de Texto (MT) é o processo de extração de informações úteis a partir de dados textuais não estruturados, transformando-os em um formato estruturado para análise posterior. Esse processo combina técnicas de Processamento de Linguagem Natural (PLN), aprendizado de máquina e estatística, permitindo identificar padrões, tendências e relações semânticas nos textos (IBM, 2025a). A mineração de texto pode ser aplicada em diversos contextos, como análise de sentimentos, detecção de fraudes, extração de conhecimento de artigos científicos e monitoramento de redes sociais.

O processo de Mineração de Texto pode ser dividido em cinco etapas principais, conforme ilustrado na Figura 2 (ARANHA, 2007).

[4] DATALOGIC developer portal - https://developer.datalogic.com/mobile-computers/news/android-rfp

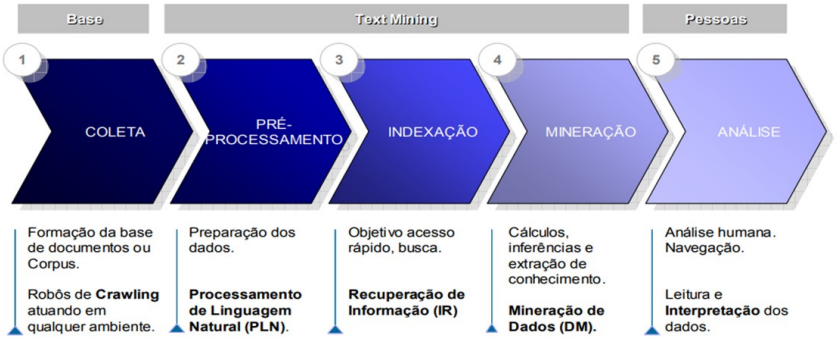

Figura 2 – Etapas do processo de Mineração de Texto. Fonte: (ARANHA, 2007).

As seções a seguir apresentam brevemente alguns detalhes das etapas de mineração destacadas na Figura 2.

2.2.1 Coleta de Dados (Web Scraping e APIs)

A Coleta de Dados consiste na obtenção dos documentos textuais que serão usados nas etapas posteriores do processo de mineração. Esses documentos podem ser coletados a partir de diversas fontes, como arquivos locais com textos armazenados em formatos como .txt, .csv, .json e .xml, ou a partir de bases de dados públicas ou privadas com documentos de interesse. Também é possível obter-se documentos a partir de serviços online como *X* (antigo Twitter), *Google Scholar* e mesmo o *YouTube*, que oferecem APIs para obtenção de dados textuais.

O *método de coleta* pode ser manual, semiautomático ou automático. Na *coleta manual*, um profissional escolhe e coleta os documentos textuais relevantes para a tarefa a ser executada. A *coleta semiautomática* envolve o uso de ferramentas e consultas a bases de documentos, porém intermediada por profissionais. Já a *coleta automática* se baseia no uso de *web crawlers* para coleta automática (podendo ser contínua ou não) de documentos a partir de bases previamente escolhidas.

Por fim, destacamos o *Web Scraping*, uma técnica automatizada de extração de textos em páginas web. Essa técnica foi utilizada na pesquisa relatada nesta dissertação e será melhor detalhada na seção a seguir.

2.2.1.1 Web Scraping na Mineração de Texto

Web Scraping (raspagem da web ou ainda *scraping* de dados) é um método automático de mineração que permite a extração automatizada de grandes quantidades de dados de sites, convertendo-os em informações estruturadas para análise posterior. Esse método utiliza softwares que simulam a navegação humana para extrair informações específicas (ZHAO, 2017). Assim, ele desempenha um papel essencial na coleta automatizada de dados da web, possibilitando a extração de textos que não estão disponíveis por meio de APIs ou bases de dados abertas.

Web Scraping é muito relevante na segurança digital, permitindo a coleta de dados sobre vulnerabilidades diretamente de fontes como bancos de dados de segurança, fóruns de especialistas e repositórios de *exploits*.

Web Scraping pode ser realizada utilizando técnicas variadas, incluindo (ZHAO, 2017):

- **Análise de DOM (Document Object Model)[5]:** Identificação de elementos HTML relevantes.
- **Expressões Regulares[6]:** Uso de padrões para localizar trechos específicos do texto.
- **Automação de Navegação:** Simulação de interação com sites dinâmicos por meio de ferramentas como Selenium[7]. Embora originalmente projetado para testes automatizados de interface web, o Selenium também pode ser amplamente utilizado em tarefas de web scraping, pois permite capturar conteúdos gerados dinamicamente em páginas que dependem de JavaScript para exibir informações.

As principais bibliotecas utilizadas para Web Scraping incluem:

- **BeautifulSoup[8]:** Parsing e extração de dados de HTML e XML.
- **Scrapy[9]:** Framework avançado para coleta de grandes volumes de dados.
- **Selenium:** Simulação de ações humanas em navegadores.

[5] https://dom.spec.whatwg.org/#what
[6] https://regexr.com/
[7] https://www.selenium.dev/
[8] https://pypi.org/project/beautifulsoup4/
[9] https://www.scrapy.org/

Essa abordagem de coleta de dados também enfrenta desafios e restrições. A restrição mais comum é feita por *bloqueio de sites* - alguns sites implementam mecanismos de proteção, como CAPTCHAs e restrições de IP. É importante ainda observar os aspectos legais e éticos na coleta de dados, uma vez que a coleta de dados sensíveis sem permissão pode violar termos de uso e leis de proteção de dados.

2.2.2 Pré-processamento dos Textos

Após a coleta, os textos precisam ser limpos e normalizados para garantir que apenas informações relevantes sejam processadas (IBM, 2025b). Esta etapa da MT, também conhecida como *preparação dos dados*, recebe como entrada o documento original e retorna uma representação mais apropriada para o tratamento automático subsequente. Essa representação de saída pode ser o texto completo pré-processado, ou uma lista (*array*) com as palavras mais relevantes para representar o texto de entrada. Essa escolha depende do objetivo do sistema de mineração de texto sendo criado.

O texto original pode passar por diversas fases de pré-processamento, que serão escolhidas de acordo com o objetivo final do sistema. Cada fase pode utilizar técnicas diferentes, sendo algumas dessas técnicas apenas baseadas em casamento de padrão, enquanto outras mais sofisticadas utilizam métodos da área de Processamento de Linguagem Natural (JURAFSKY; MARTIN, 2009). Destacamos a seguir as principais fases de pré-processamento nesta etapa.

- **Tokenização:** Separação do texto em unidades menores (*tokens*), como palavras, sinais de pontuação, numerais e caracteres especiais, resultando em uma lista de tokens passada para a próxima fase.

- **Normalização:** Padronização de caracteres (geralmente, letras em caixa baixa), remoção de acentos e caracteres especiais. O objetivo aqui é facilitar o casamento de padrão entre palavras no processo geral de mineração.

- **Remoção de Stopwords:** Eliminação de palavras sem valor semântico (artigos, preposições e conjunções), palavras frequentes sem relevância para o processo de mineração sendo executado.

- **Lematização:** Redução da palavra à sua forma base, seu lema (e.g., "correndo"$\rightarrow$ "correr"), com o objetivo de aglutinar em um único token as palavras flexionadas derivadas

de uma mesma raiz.

- **Stemming:** Redução da palavra ao seu "*stem*"(e.g., "correndo"→ "corr"). Esta técnica tem o mesmo objetivo da lematização, porém adota um algoritmo mais simples que apenas corta os sufixos das palavras, sem de fato reduzi-las a um radical morfológico. Por ser mais simples de implementar, esta tem sido a escolha na maioria dos sistemas.

Podemos citar ainda outras técnicas importantes nessa etapa da mineração, porém que não são utilizadas com tanta frequência nos trabalhos de PLN e MT.

- **Correção ortográfica** automática do texto utilizando-se bibliotecas e dicionários já disponíveis digitalmente. Esta fase também auxilia o processo de casamento de padrão entre palavras, por eliminar erros de ortografia que iriam distinguir palavras iguais.
- **Uso de dicionários de sinônimos** (tesauros) digitalizados para ampliar ou restringir automaticamente o vocabulário a ser considerado no processo de mineração, sendo útil em algumas atividades, como por exemplo na recuperação de documentos indexados.
- **Etiquetagem automática da classe gramatical** das palavras (do inglês, *Parts-Of-Speech tagging* - **POS-tagging**), muito útil em sistemas de mineração de opinião (uma vez que opiniões geralmente utilizam adjetivos e advérbios).
- **Identificação automática de entidades nomeadas**, tais como nomes de pessoas, instituições, localidades (cidades, estados, etc), entre outros. Essa atividade é útil em tarefas de extração de informação, quando os dados a serem extraídos se referem a essas entidades nomeadas.

Depois de realizar algumas das fases acima, o documento passa a ser representado por uma lista de tokens na forma que foi determinada pelas técnicas utilizadas no pré-processamento. Por exemplo, se a operação de *lematização* foi realizada, os tokens serão *lemas* do idioma do documento original. A seguir, alguns sistemas criam uma representação vetorial do documento para possibilitar tarefas futuras (seção a seguir).

2.2.2.1 Representação Vetorial do Texto

Para facilitar a análise posterior dos textos de entrada, alguns sistemas de mineração criam representações estruturadas em forma vetorial (BAEZA-YATES; RIBEIRO-NETO, 2013).

Essas representações são criadas a partir da lista de tokens (palavras que podem ter sofrido modificações como retirada de acentuação ou *stemming*) que resultaram da execução das fases vistas acima.

A representação mais simples utilizada é conhecida como "sacola de palavras" – do inglês, *Bag of Words (BoW)*. Nessa lista cada token só aparece uma vez, juntamente com a informação da sua frequência de ocorrência no documento original (seu "peso" ou importância para representar o documento). Essa representação é mais concisa do que a lista gerada na etapa anterior, porém ela perde informações da sequência original das palavras e, em consequência, informações da sintaxe da frase.

Apesar da sua simplicidade, essa representação é muito usada em tarefas de mineração referentes a indexação e recuperação de documentos, onde os documentos são ranqueados pelos pesos das suas palavras que aparecem na consulta (BAEZA-YATES; RIBEIRO-NETO, 2013), bem como de classificação e agrupamento, onde os pesos são usados para identificar documentos de uma dada classe ou para agrupar os documentos semelhantes (JURAFSKY; MARTIN, 2009).

Contudo, nem sempre a performance desses sistemas é satisfatória considerando apenas os pesos das palavras em cada documento. Por exemplo, se a palavra "mineraçãosacola de tem alta frequência de ocorrência em todos os documentos indexados pelo sistema, ou na base de documentos que serão classificados, esse token não vai ajudar no ranking nem na tarefa de classificação (não será um bom discriminante de classe).

Como solução, podemos eliminar das BoWs as palavras muito frequentes em todos os documentos da base (seriam consideradas *stopwords*). Isso é muito comum nas tarefas de classificação de texto. Porém, pode não ser a solução para indexação de documentos, pois uma consulta com a palavra "mineração" não irá recuperar nenhum documento da base.

A ideia então é utilizar uma fórmula que calcula os pesos dos tokens considerando também sua frequência na base de documentos sendo processados. A fórmula mais utilizada para esse cálculo é conhecida como *TF-IDF (Term Frequency-Inverse Document Frequency)*, que calcula a relevância de cada token considerando sua frequência no documento, porém penalizando tokens que ocorrem em muitos documentos da base (BAEZA-YATES; RIBEIRO-NETO, 2013). Essa fórmula de cálculo de pesos melhora o desempenho de classificadores de texto e de sistemas de recuperação de informação.

Apesar de muito usados, os modelos baseados em listas de tokens com pesos associados sofrem a limitação de não considerarem a semântica das palavras. Nesse cenário surgem os modelos vetoriais mais sofisticados, que buscam capturar relações semânticas entre as palavras

dos textos, conhecidos como *Word Embeddings* (JURAFSKY; MARTIN, 2025; IBM, 2025c).

Word embeddings são um modo de representar palavras como vetores em um espaço vetorial multidimensional no qual cada eixo representa um "conceito" (IBM, 2025c). Cada palavra do texto sendo processado é representada por um vetor cuja direção é determinada pela sua relação com os conceitos representados nos eixos do espaço vetorial. Nesse espaço vetorial, a distância e a direção entre vetores refletem a *similaridade* e a *relação* entre as palavras representadas (ver Figura 3 como exemplo[10]).

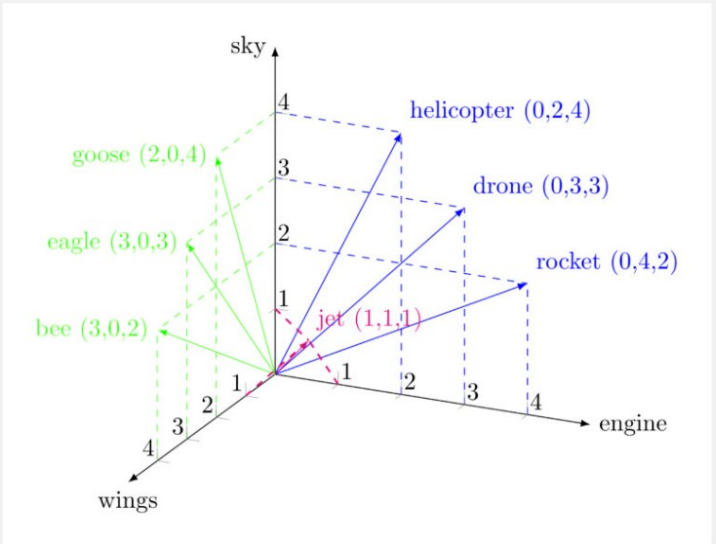

Figura 3 – Exemplo de *Word embedding*. Fonte: <https://corpling.hypotheses.org/495>

Esse exemplo considera um conjunto de apenas sete palavras: abelha, águia, ganso, helicóptero, drone, foguete e jato (do inglês, respectivamente, *bee, eagle, goose, helicopter, drone, rocket, jet*). Aqui, três contextos foram considerados: asas, motor e céu (do inglês, respectivamente, *wings, engine, sky*). Cada contexto considerado será representado por um eixo do espaço cartesiano, e os termos são representados por vetores cuja direção é determinada por suas coordenadas (x,y,z) no espaço cartesiano. O valor das coordenadas de cada termo corresponde ao número de vezes que o termo foi encontrado em cada contexto.

Por exemplo, helicópteros não possuem asas, assim é natural que esse termo não seja encontrado nesse contexto específico (eixo z). Já o objeto "drone" está relacionado aos conceitos

[10] https://corpling.hypotheses.org/495

de "céu" e "máquina", porém não tem relação com o conceito "asas". Por fim, o objeto "jato" se relaciona com os três conceitos constantes desse espaço vetorial simplificado.

A ideia base dos *embeddings* é que a similaridade semântica entre termos pode ser representada por contextos afins. Nesse exemplo, "helicóptero" e "drone" são próximos porque ocorrem em contextos similares. Assim, essa representação pode ser usada para medir *similaridade semântica* entre termos observando-se a distância angular entre os vetores que representam cada termo. Modelos como Word2Vec (MIKOLOV et al., 2013), FastText (ATHIWARATKUN; WILSON; ANANDKUMAR, 2018) e BERT (DEVLIN et al., 2019) são largamente utilizados para facilitar a construção e o uso de representações baseadas em *Word embeddings* (JOHNSON; MURTY; NAVAKANTH, 2024).

2.2.3 Indexação e Recuperação automática

Esta etapa dos sistemas de mineração tem por objetivo indexar os documentos da base sendo considerada, a fim de facilitar a recuperação de documentos relevantes a partir de consultas do usuário. Note que esta etapa é opcional para sistemas de mineração de texto e só é implementada quando necessário – essa escolha vai depender dos objetivos do sistema.[11]

Esta etapa objetiva a construção de um "engenho de busca" utilizando técnicas das áreas de PLN e de Recuperação de Informação (RI). RI pode ser definida como a área de pesquisa e desenvolvimento que investiga métodos e técnicas para a representação, a organização, o armazenamento, a busca e a recuperação de itens de informação (documentos) com o objetivo principal de facilitar o acesso a documentos relevantes à necessidade de informação do usuário, geralmente representada através de consultas baseadas em palavras-chaves (BAEZA-YATES; RIBEIRO-NETO, 2013).

Os sistemas que implementam engenhos de busca contam com duas fases principais: (1) Criação da Base de índices de documentos; (2) Consulta à Base de índices de documentos.

A Fase 1, *Criação da Base de índices*, conta com três passos principais:

- Aquisição (seleção) dos documentos a serem indexados
- Pré-processamento (preparação) dos documentos

[11] Esta etapa não foi implementada no sistema protótipo desenvolvido neste trabalho de mestrado por não ter se mostrado necessária.

- Indexação dos documentos, que consiste na criação da *base de índices invertidos*. Esta é uma estrutura na qual as palavras apontam para os índices documentos onde elas ocorrem, facilitando assim a fase de recuperação de documentos. Esses índices são os identificadores dos documentos de entrada, podendo ser endereços locais ou URLs, por exemplo.

A Fase 2, *Consulta à Base de índices*, conta com 4 passos principais:

- Construção da consulta (*query*), realizada pelo usuário. Note que as consultas serão pré-processadas como acontece na Fase 1 com os documentos indexados, a fim de possibilitar o casamento (*match*) entre os tokens criados na Fase 1 e as palavras digitadas na consulta.

- Busca (casamento/*match* com a consulta do usuário), onde o algoritmo implementado recupera os identificadores dos documentos que contêm com as palavras digitadas na consulta.

- Ordenação dos documentos recuperados, onde a lista de índices será ordenada de acordo com os critérios implementados no sistema de RI sendo usado.

- Apresentação dos resultados, que mostra na interface do usuário a lista de identificadores ordenados por sua relevância em relação à consulta do usuário.

2.2.4 Mineração de Dados/Informação

A etapa de *mineração de dados/informação* é a mais variada, objetivando extrair conhecimento novo a partir dos documentos sendo analisados. Essa etapa pode cobrir várias tarefas, dependendo do foco do processo de mineração a ser realizado. As técnicas utilizadas nessas tarefas são oriundas de diversas áreas da Computação, tais como Aprendizagem de Máquina (AM), raciocínio baseado em regras, PLN e RI. Algumas técnicas são baseadas em casamento de padrões, enquanto outras utilizam processos mais sofisticados de mineração. Destacamos a seguir as tarefas mais comuns nessa etapa.

A *Classificação* ou *Categorização* de texto visa associar documentos a classes pré-definidas - e.g., email *spam* ou *não-spam*, notícias de *esporte, economia* ou *cultura* (KOWSARI et al., 2019). Os documentos são classificados a partir de características do texto, como termos

ou palavras presentes nos documentos. Os textos de entrada são pré-processados, sendo geralmente representados por vetores de palavras BoW com pesos calculados usando TF-IDF. Os principais algoritmos de aprendizagem de máquina utilizados nesta tarefa são K-nearest neighbors (KNN), Naïve Bayes, Árvores de Decisão, Redes Neurais e Support Vector Machine (SVM).

A tarefa de *Agrupamento* (do inglês, *clustering*) difere um pouco da classificação por não ter classes pré-definidas (AGGARWAL; ZHAI, 2012). O processo recebe a base de documentos pré-processados e identifica similaridades entre os textos, criando grupos que contenham documentos semelhantes entre si, e que sejam diferentes dos documentos nos outros grupos. Os grupos (clusters) podem ser de um nível (*flat*) ou hierárquicos. Alguns algoritmos usados para isto são o K-Means (*flat*) ou DBSCAN.

A *Extração de Informação* tem por objetivo extrair e estruturar informações específicas relevantes para o usuário a partir de documentos textuais (GRISHMAN, 2015; XU et al., 2024). Distinguimos duas tarefas de extração: *extração baseada em formulário* e *extração aberta*. No primeiro caso, os documentos de entrada estão associados a um domínio de aplicação específico, e os dados a serem extraídos são previamente definidos em um *template* (formulário). Geralmente, a extração é realizada utilizando técnicas de casamento de padrões através de Expressões Regulares (RegEx). Já a *extração aberta* recebe como entrada um documento qualquer e busca extrair relações (em forma de tuplas) não determinadas a priori (KAMP et al., 2023). Aqui as técnicas utilizadas são mais sofisticadas, sendo oriundas das áreas de PLN e Aprendizagem de Máquina.

A *Análise de Sentimentos/Polaridade* (ou *Mineração de Opinião*) tem por objetivo identificar a polaridade da opinião (o "sentimento") dos usuários a respeito de alguma entidade, como: um produto específico, uma empresa, um local, um evento ou uma pessoa. É uma área de grande importância devido ao crescimento de opiniões dispersas na Web, sendo útil nas áreas de marketing em geral, de serviços, de avaliação de popularidade, entre outras. Aqui as técnicas predominantes são AM e PLN (WANKHADE; RAO; KULKARNI, 2022).

Sumarização de texto visa identificar as informações mais importantes de um texto para produzir uma versão resumida desse texto, porém que guarde as informações mais relevantes do texto original. Atualmente essa tarefa tem sido realizada via uso de Modelos de Linguagem, com resultados superiores aos algoritmos usados anteriormente (SCHAIK; PUGH, 2024).

Extração de regras de associação visa identificar padrões presentes em um documento específico (por exemplo, se os termos "ações" e "empresa" aparecem no documento, o termo

"Bolsa de valores" provavelmente também aparece) ou padrões em uma mesma base de documentos (por exemplo, páginas Web consultadas em sequência, tais como compra de passagens e reserva de hotel).

2.2.4.1 Medidas de Similaridade Textual

Esta seção se dedica a apresentar uma tarefa mais específica da Mineração de Texto, que consiste em mensurar a similaridade entre textos (PRAKOSO; ABDI; AMRIT, 2021). Esse tema é muito relevante para nosso trabalho, uma vez que devemos avaliar quando dois relatos distintos tratam da mesma vulnerabilidade, a fim de identificar redundâncias (ver Seção 4.4.1) do Capítulo 4).

De acordo com (PRAKOSO; ABDI; AMRIT, 2021), os métodos para medir similaridade textual podem ser classificados em cinco categorias: baseados em strings, baseados em corpus, baseados em conhecimento, e híbridos. Dentre os métodos citados, usamos no nosso trabalho um método baseado em strings e um método específico baseado em grandes corpora, os *Word Embeddings*.

2.2.4.1.1 Métodos Baseados em Strings

Métodos baseados em strings são os mais tradicionais e mais simples de implementar, e avaliam a similaridade entre duas *strings* (no nosso caso, dois *textos*) comparando a sequência de *tokens* (no nosso caso, *palavras*) de cada string. Esses métodos geralmente avaliam a similaridade com base na *distância de edição*[12] entre duas strings, calculando o número mínimo de operações necessárias para transformar uma dada string em outra. Assim, quanto maior a distância de edição entre duas strings, menor será a similaridade entre elas. Por exemplo, a distância de edição entre "o carro vermelho" e "o carro" seria 1, pois basta excluir a palavra "vermelho" para igualar as duas strings. Já a distância entre "o carro vermelho" e "carro preto" seria 3, pois duas palavras foram excluídas e uma foi inserida.

Como exemplo mais relevante dessa categoria de método, temos a *Distância de Levenshtein* (LEVENSHTEIN, 1966)[13]. Essa medida, apesar de antiga, ainda é usada com muito sucesso no PLN, tendo inspirado variações mais refinadas. Outro método muito utilizado é a subsequência

[12] https://en.wikipedia.org/wiki/Edit_distance
[13] https://en.wikipedia.org/wiki/Levenshtein_distance

comum mais longa (AZIZ, 1965)[14]. As operações de edição consideradas variam entre os métodos existentes. Por exemplo, a Distância de Levenshtein considera inserções, exclusões e substituições de tokens. Já a subsequência comum mais longa só considera inserções e exclusões.

Apesar da grande utilidade e facilidade de uso desses métodos, eles têm uma limitação quando usados para medir distância entre documentos: eles consideram apenas a sequência de palavras nos textos sendo comparados, não levando em conta o significado das palavras.

2.2.4.1.2 Métodos Baseados em Grandes Corpora

Métodos baseados em grandes corpora utilizam um corpus externo já existente para extrair a relação entre palavras ou textos. Alguns métodos aferem inicialmente a relação entre palavras, e em seguida usam esses valores para avaliar a similaridade no nível de frase. Por exemplo, palavras diferentes que ocorrem muitas vezes em um mesmo contexto podem ter alto grau de similaridade (e.g., "o *carro* é veloz" e "o *automóvel* é veloz") (JURAFSKY; MARTIN, 2025). Outros métodos mais sofisticados podem medir a similaridade de texto diretamente, sem passar pelo nível da palavra.

Ainda dentro dessa abordagem de grandes corpora, destacamos o uso de representações baseadas em *Word Embeddings* para cálculo de similaridade textual. Como visto na Seção 2.2.2.1, *word embeddings* são representações vetoriais n-dimensionais que tentam capturar o significado das palavras relacionando-as a conceitos semânticos. Essas representações são obtidas através da aplicação de métodos de aprendizagem profunda (*deep learning*) a grandes corpora de textos. Como já mencionado, modelos como Word2Vec e BERT são largamente utilizados para facilitar a construção e o uso de representações baseadas em *Word embeddings* (JOHNSON; MURTY; NAVAKANTH, 2024).

De fato, os *embeddings* representam palavras como vetores numéricos. A distância entre os vetores das palavras incorporadas é considerada como semanticamente significativa. Para medir a similaridade entre duas frases, o método representa as frases como vetores normalizados de palavras. A distância entre as duas frases é medida usando uma função que calcula a distância vetorial entre as palavras da primeira frase e as palavras da segunda frase. Assim, quanto maior a distância entre as duas frases, menor a similaridade entre elas.

[14] https://en.wikipedia.org/wiki/Longest_common_subsequence

Dentre as opções dentro desse método, destacamos aqui o *SBERT* (*Sentence-BERT*[15]), utilizado neste trabalho para capturar similaridade semântica entre textos. SBERT transforma frases em vetores numéricos, permitindo comparações mais precisas, baseadas no significado das palavras. Assim, palavras sinônimas ou fortemente relacionadas terão alta similaridade dentro desse modelo.

2.2.4.1.3 Métodos Baseados em Conhecimento

Métodos baseados em conhecimento utilizam redes de conceitos/termos semanticamente relacionados para medir a similaridade entre palavras, e depois então avaliar a similaridade no nível de frase. Existem diferentes métodos para avaliar a relação entre as palavras, sendo comum considerar o comprimento do caminho percorrido para chegar de um conceito ao outro na rede como sendo a similaridade entre os conceitos. A rede semântica pode ser específica para um dado domínio, como medicina ou direito, ou podem ser de uso geral, como a WordNet[16]. Contudo, redes genéricas nem sempre oferecem bons resultados, devido à ambiguidade das línguas naturais. Então resta a dificuldade de produzir uma rede específica para cada domínio, bem como de extrapolar a similaridade entre palavras para o nível da frase.

2.2.4.1.4 Métodos Híbridos

Já os métodos híbridos combinam dois ou mais métodos existentes para usufruir das vantagens individuais de cada método, obtendo soluções mais precisas para medir similaridade. Por exemplo, o método proposto por (LI et al., 2006) para calcular a similaridade de frases considera informações semânticas e a ordem de palavras nas frases. Para calcular o significado semântico das frases, eles combinaram um método baseado em conhecimento e um método baseado em corpus.

2.2.5 Análise dos Resultados (Avaliação)

A última etapa de mineração consiste na análise e na avaliação dos resultados obtidos, sendo essa etapa fundamental para garantir que os padrões identificados sejam significativos e úteis para os processos de tomada de decisão.

[15] SBERT - https://sbert.net/
[16] https://wordnet.princeton.edu/

A avaliação é realizada por um *analista de dados*, que fica responsável por interpretar os resultados obtidos pela etapa de mineração. Geralmente, trata-se de uma atividade parcialmente automática (uso das métricas quantitativas) e parcialmente manual (análise qualitativa dos resultados).

2.2.5.1 Métricas Computacionais

Na fase de análise automática o analista aplica métricas computacionais quantitativas para avaliar a eficácia dos modelos e algoritmos utilizados. Essas métricas variam de acordo com a tarefa principal que foi foco do processo de mineração.

Principais Métricas Computacionais (BAEZA-YATES; RIBEIRO-NETO, 2013):

- **Precisão (*Precision*):** Mede a proporção de predições corretas entre as retornadas.
- **Revocação (*Recall*):** Mede a proporção de predições corretamente detectadas entre as existentes na base de dados.
- **F1-Score:** Média harmônica entre precisão e revocação.
- **Tempo de Execução:** Avaliação da eficiência do processamento.

2.2.5.2 Validação Qualitativa

Por fim, o analista de dados realiza uma *validação qualitativa*, considerando a relevância e a coerência dos resultados obtidos. Abaixo citamos algumas técnicas e métodos de visualização de dados mais comuns:

- **Mapas de calor (Heatmaps)**[17]**:** Técnica gráfica que utiliza cores para representar a importância de cada dado dentro de um conjunto de objetos. Heatmaps de similaridade utilizam cores para representar a similaridade entre objetos, sendo apresentados frequentemente em forma de matriz, onde as cores indicam a proximidade ou diferença entre os elementos comparados.
- **Nuvens de palavras (Word Clouds)**[18]**:** Representação visual do texto, onde o tamanho das palavras é determinado pela sua frequência de ocorrência no corpus. Essa técnica é

[17] https://en.wikipedia.org/wiki/Heat_map
[18] https://www.wordclouds.com/

útil para destacar as palavras mais comuns ou importantes dentro de um conjunto de dados textuais.

- **Gráficos de Dispersão**[19]**:** Os gráficos convencionais representam a relação entre duas variáveis quantitativas através de pontos em um plano cartesiano no qual cada eixo representa o valor de uma das variáveis. Por exemplo, cada ponto no gráfico pode representar um funcionário de uma empresa, e as variáveis podem ser "horas extras trabalhadas (x)" e "adoecimento dos trabalhadores (y)". Assim, o gráfico facilita a visualização da relação entre essas variáveis, além da detecção de padrões e tendências nos dados.

- **Gráficos de Dispersão Textual:** Similar aos gráficos de dispersão convencionais, porém aplicados a dados textuais. Neste caso, cada ponto do gráfico representa um documento de um conjunto previamente escolhido, permitindo uma visualização clara da distribuição e do relacionamento entre os dados. Esses gráficos podem ser úteis em tarefas de classificação e agrupamento. Vale salientar que é necessário usar algum método para transformar os textos em pontos.

- **Gráficos de Dispersão Textual baseada em Word Embeddings**[20]**:** No nosso caso específico, o objetivo é visualizar a similaridade entre documentos com base na relação semântica entre suas palavras ou sentenças. Nesses casos, os documentos são inicialmente representados como vetores através do uso de word embreddings (e.g., SBERT), e em seguida são transformados em pontos pelo uso da técnica de Análise de Componente Principal (do inglês, *Principal Component Analysis* - PCA[21]). A proximidade dos pontos no gráfico indica maior semelhança semântica entre os textos representados.

As técnicas de análise e visualização de dados aqui mencionadas foram utilizadas na avaliação do nosso trabalho, e serão então exemplificadas no Capítulo 5.

[19] https://www.fm2s.com.br/blog/grafico-de-dispersao

[20] <https://medium.com/data-science/text-embeddings-comprehensive-guide-afd97fce8fb5>, <https://programminghistorian.org/en/lessons/clustering-visualizing-word-embeddings>

[21] PCA é uma técnica de redução linear de dimensionalidade utilizada na análise exploratória de dados, visualização e processamento de dados - <https://en.wikipedia.org/wiki/Principal_component_analysis>

2.3 CONSIDERAÇÕES FINAIS

Este capítulo apresentou alguns conceitos básicos e definições em áreas relacionadas ao tema da pesquisa relatada neste documento.

A Seção 2.1 teve como foco a área de Teste de Software, com destaque para as seções 2.1.3 e 2.1.3.1, que tratam de vulnerabilidades de software. A seguir, a Seção 2.2 se concentrou na grande área de Mineração de Textos, apresentando seus processos principais. Aqui destacamos três seções de maior importância para a pesquisa realizada objetivando tratar relatos sobre vulnerabilidades: Seção 2.2.1.1, sobre Web Scraping; Seção 2.2.2, sobre Pré-processamento de Texto; e Seção 2.2.4.1, sobre Medidas de Similaridade Textual.

A seguir, o Capítulo 3 aprofunda o tema central da nossa pesquisa, a identificação de vulnerabilidades em SOs para dispositivos móveis. Serão apresentados os principais SOs para smartphones, seguindo com uma apresentação mais detalhada sobre suas vulnerabilidades. Por fim, veremos alguns trabalhos relacionados ao tema central desta pesquisa.

3 IDENTIFICAÇÃO E TRATAMENTO DE VULNERABILIDADES DE SOS PARA DISPOSITIVOS MÓVEIS

Este capítulo aborda a identificação de vulnerabilidades em sistemas operacionais para dispositivos móveis, bem como algumas soluções existentes na atualidade para prevenir e/ou corrigir tais problemas.

Inicialmente, a Seção 3.1 traz uma breve apresentação dos SOs mais utilizados em dispositivos móveis, o iOS e o Android, descrevendo sua arquitetura e as implicações na segurança dos dispositivos. A seguir, a Seção 3.2 trata especificamente das vulnerabilidades de tais sistemas, enumerando os tipos mais comuns, erros de código que implicam em quebra de segurança e métodos de detecção desses problemas.

A Seção 3.3 aborda relatos disponíveis na Web descrevendo vulnerabilidades. Tais relatos podem ser encontrados em páginas especializadas com descrições textuais, bem como em canais do YouTube. Esses relatos trazem pistas relevantes na detecção e correção de defeitos dos SOs, sendo um importante complemento aos testes de SW geralmente conduzidos pelas empresas.

A Seção 3.4 apresenta alguns trabalhos relacionados à identificação de vulnerabilidades no Android. Como não encontramos trabalhos que também exploram relatos na Web, limitamos a apresentação a pesquisas desenvolvidas utilizando as abordagens tradicionais de teste de software. Por fim, a Seção 3.5 traz as considerações finais deste capítulo.

3.1 SISTEMAS OPERACIONAIS EM DISPOSITIVOS MÓVEIS

Esta seção apresenta brevemente os dois sistemas operacionais mais populares para dispositivos móveis: iOS e Android. Como o foco da nossa pesquisa é no Android, esse SO foi melhor detalhado aqui (Seção 3.1.2).

3.1.1 iOS (Apple)

O sistema operacional iOS da Apple[1] é amplamente reconhecido por suas robustas medidas de segurança, resultantes de uma arquitetura bem projetada e de uma abordagem voltada à privacidade. No entanto, como qualquer sistema, ele não é imune a vulnerabilidades e ataques.

[1] https://www.apple.com/br/ios/

Aqui estão algumas das principais características de segurança do iOS, como descrito nas páginas de suporte da Apple[2]:

- **Modelo de Atualização Centralizado**

  Uma das principais vantagens do iOS é seu modelo de atualização centralizado. A Apple controla rigidamente o processo de distribuição de atualizações de software, o que permite a rápida implementação de *patches* de segurança para todos os dispositivos compatíveis. Isso reduz significativamente a janela de exposição a vulnerabilidades conhecidas, uma vez que os usuários recebem atualizações simultaneamente[3].

- **Arquitetura de Segurança em Camadas**

  O iOS utiliza uma arquitetura de segurança em camadas para proteger os dispositivos e os dados dos usuários:

  - **Kernel XNU** - No núcleo do sistema está o kernel XNU, que combina elementos dos kernels Mach e BSD. Ele é responsável por gerenciar recursos e garantir a segurança a nível de sistema operacional, isolando processos e limitando privilégios[4].
  - **Security Enclave:** Os dispositivos modernos da Apple incluem um coprocessador *Secure Enclave*, que gerencia dados sensíveis, como autenticações biométricas, de forma segura e isolada do resto do sistema[5].
  - **Sandboxing:** Todos os aplicativos no iOS operam em um ambiente de *sandbox*, isolando-os uns dos outros e do sistema principal. Isso impede que um aplicativo malicioso comprometa outros aplicativos ou o próprio sistema operacional[6].

- **Controle da *App Store***

  A Apple exerce um controle rigoroso sobre sua *App Store*, revisando e aprovando cada aplicativo antes que ele esteja disponível para download. Esse processo de revisão ajuda a minimizar a disseminação de aplicativos maliciosos e protege os usuários contra softwares potencialmente prejudiciais[7].

[2] https://support.apple.com/pt-br/guide/
[3] https://support.apple.com/pt-br/guide/deployment/depc4c80847a/web
[4] https://support.apple.com/pt-br/guide/security/welcome/web
[5] https://support.apple.com/pt-br/guide/security/sec59b0b31ff/1/web/1
[6] https://support.apple.com/pt-br/guide/deployment/depe1553f932/web
[7] https://support.apple.com/pt-br/guide/security/secb8f887a15/web

- **Desafios de Segurança**

  Embora o iOS seja menos suscetível a ataques em comparação ao Android, ele ainda enfrenta questões de segurança. Recentemente, foi descoberta uma vulnerabilidade nos chips *M-series* que permite a extração de chaves de criptografia, comprometendo a segurança de software criptográfico. Essa falha não pode ser corrigida diretamente, exigindo mitigações específicas em softwares de terceiros, o que pode resultar em degradação de desempenho[8].

3.1.2 Android

A arquitetura do sistema operacional Android é estruturada em camadas, o que influencia diretamente as implicações de segurança associadas a cada componente. Essa estrutura em camadas é projetada para possibilitar modularidade e abstração de processos, mas também apresenta desafios de segurança específicos. A seguir, temos uma breve apresentação do SO Android, comentando sobre suas características principais. A Figura 4 traz uma imagem da pilha de software do Android segundo o site *Android Open Source Project* [9].

- **Camadas da Arquitetura e Implicações de Segurança do Android**[10]

  - **Kernel Linux:** Na base da arquitetura do Android está o *Kernel Linux*[11], que gerencia o hardware do dispositivo, incluindo drivers e gerenciamento de memória. O kernel é crítico para a segurança, pois qualquer vulnerabilidade aqui pode permitir acesso privilegiado ao dispositivo, comprometendo todo o sistema.
  - **Camada de Abstração de Hardware (*Hardware Abstraction Layer* - HAL):** Facilita a comunicação entre a camada superior do Android e o hardware do dispositivo através de APIs padronizadas. Vulnerabilidades na HAL podem possibilitar manipulações diretas do hardware e dos softwares por aplicativos maliciosos.
  - **Middleware:** Acima do kernel está o middleware, que inclui as bibliotecas nativas em C/C++ que fornecem funcionalidades essenciais, como gráficos e bancos de dados, e também a máquina virtual *Android Runtime* (ART) ou *Dalvik*, dependendo

[8] https://www.macrumors.com/2024/03/22/apple-silicon-vulnerability-encryption-keys/
[9] https://source.android.com/docs/security/overview?hl=pt-br#background
[10] https://source.android.com/docs/core/architecture?hl=pt-br
[11] https://www.linux.org/

da versão do Android. ART e Dalvik são responsáveis pela execução de aplicativos, onde cada aplicativo (Java) é executado em sua própria instância para isolamento. Falhas nessas bibliotecas podem levar a vulnerabilidades críticas, como a execução de código arbitrário.

– **Framework de Aplicações Java:** Esta camada fornece as APIs que os desenvolvedores utilizam para criar aplicativos. Inclui serviços do sistema como o gerenciador de atividades e serviços de localização. A segurança do framework depende da implementação correta das APIs e do gerenciamento de permissões pelos desenvolvedores.

– **Aplicações:** No topo da arquitetura estão os aplicativos, que interagem com o framework para fornecer funcionalidades ao usuário. Essa camada é frequentemente alvo de ataques, dado que os aplicativos podem solicitar e obter permissões para acessar dados sensíveis, sendo a camada mais vulnerável devido ao foco dos desenvolvedores.

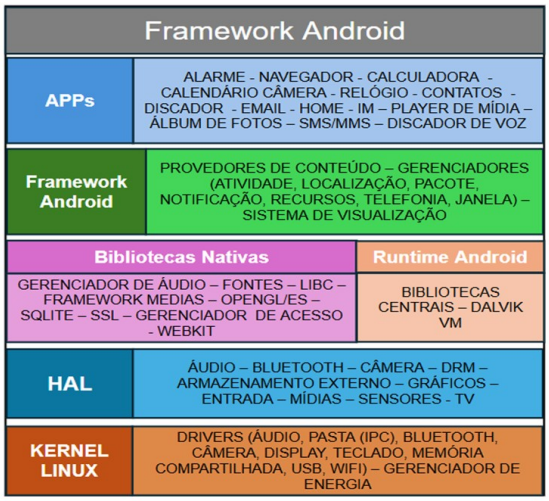

Figura 4 – Pilha de software do Android

Fonte: <https://source.android.com/docs/security/overview?hl=pt-br#background>

O Android implementa vários mecanismos de segurança para proteger o sistema e os dados dos usuários. Abaixo destacamos alguns desses importantes mecanismos em uso.

- **Mecanismos de Segurança do Android**[12] **(ENCK et al., 2014)**

[12] <https://source.android.com/docs/security/overview>

  - **Sandboxing de Processos:** Cada aplicativo opera em um sandbox isolado. Esse design visa proteger os aplicativos de interferências entre aplicativos e o sistema. No entanto, a execução eficaz desse isolamento depende da integridade das interfaces de programação de aplicativos (do inglês, *application programming interfaces* - APIs) e do gerenciamento correto de permissões pelo usuário e pelos desenvolvedores.
  - **Assinatura Digital:** Aplicativos devem ser assinados digitalmente com uma chave privada antes de serem liberados, garantindo a integridade e a autenticidade do aplicativo.
  - **Mecanismo de Permissões:** A segurança dos aplicativos Android depende significativamente do sistema de permissões que controla o acesso dos aplicativos a funções e dados do sistema. Aplicações maliciosas podem abusar das permissões para coletar dados sensíveis ou interferir na operação de outros aplicativos. Vulnerabilidades no design de APIs ou na lógica de permissões podem levar a vazamentos de segurança significativos.
  - **Atualizações de Segurança:** O modelo de atualização do Android, devido à sua fragmentação entre diferentes fabricantes e dispositivos, frequentemente atrasa a implementação de *patches* de segurança necessários. Isso pode deixar dispositivos vulneráveis à exposição de falhas conhecidas por períodos prolongados.

- **Desafios Adicionais**

  - **Integridade de Aplicações de Terceiros:** A *Google Play Store* e outras fontes de aplicativos representam um vetor de ataque potencial. Apesar dos esforços para verificar aplicativos para detectar malware, aplicativos maliciosos às vezes escapam dessa verificação e podem ser instalados por usuários desavisados.
  - **APIs e SDKs de Terceiros:** Desenvolvedores frequentemente dependem de bibliotecas e SDKs (do inglês, *software development kits*) de terceiros para adicionar funcionalidades aos aplicativos. Esses componentes podem introduzir vulnerabilidades não intencionais se não forem devidamente seguros.

3.2 VULNERABILIDADES EM SISTEMAS OPERACIONAIS DE DISPOSITIVOS MÓVEIS

Vulnerabilidade em sistemas operacionais móveis são falhas que permitem a execução de comandos maliciosos, acesso não autorizado a dados ou interrupção dos serviços normais. Considerando o aumento no uso de dispositivos móveis, especialmente smartphones, a identificação e mitigação dessas vulnerabilidades é crucial para a segurança do usuário e a integridade do sistema.

Estudos indicam que tanto Android quanto iOS possuem suas vulnerabilidades específicas, frequentemente exploradas por ataques. Essas vão desde vulnerabilidades no kernel até falhas em aplicativos de terceiros (FARUKI et al., 2015).

Esta seção aborda vulnerabilidades em sistemas operacionais móveis, focando especialmente no Android. Essa análise abrange métodos de detecção de vulnerabilidades e técnicas para mitigar esses riscos.

3.2.1 Tipos Comuns de Vulnerabilidades

As vulnerabilidades em sistemas operacionais móveis podem ser categorizadas em várias classes, incluindo(FARUKI et al., 2015):

- **Injeções de Código:** *SQL injection* e outras formas de injeção que permitem a execução de comandos maliciosos.
- **Exposição de Dados Sensíveis:** Falhas que permitem acesso não autorizado a dados confidenciais armazenados ou transmitidos pelo aplicativo.
- **Vulnerabilidades de Criptografia:** Falhas no uso de SSL/TLS (do inglês, *Secure Sockets Layer* e *Transport Layer Security*), permitindo interceptação de dados.
- **Vulnerabilidades de WebView:** Exposição a ataques através de componentes WebView mal configurados.
- **Execução de Código Arbitrário:** Vulnerabilidades que permitem a execução de código arbitrário no dispositivo, frequentemente explorando falhas no ART ou nas bibliotecas nativas.

- **Falhas de Autenticação:** Problemas que permitem burlar mecanismos de autenticação ou a execução de ações não autorizadas.

  - **FRP Bypass:** trata-se de uma técnica ou conjunto de técnicas utilizadas para burlar a Proteção de Redefinição de Fábrica (do inglês, *Factory Redefinition Protection* - FRP) em dispositivos Android. Essas técnicas podem envolver o uso de ferramentas de software, exploração de vulnerabilidades no sistema operacional, ou manipulação de configurações de acessibilidade, entre outros métodos. O objetivo do *FRP Bypass* é desativar a proteção que exige que o usuário faça login com a conta Google previamente registrada, permitindo assim o acesso ao dispositivo sem as credenciais do usuário anterior[13].

3.2.2 Categorização de Vulnerabilidades

As vulnerabilidades em aplicativos Android são frequentemente categorizadas usando classificações conhecidas, como **CVE** (Vulnerabilidades e Exposições Comuns - do inglês, *Common Vulnerabilities and Exposures*) e **CWE** (Enumeração de Fraquezas Comuns - do inglês, *Common Weakness Enumeration*)[14]. Essa padronização na classificação auxilia no rastreio de vulnerabilidades, facilitando o envio de avisos sobre o que foi identificado, dando mais velocidade e confiabilidade a esse trabalho.

- **CVE:** Utiliza um identificador único para vulnerabilidades conhecidas. Oferece uma lista de ameaças de segurança categorizadas dentro de um sistema de referência padronizado. O programa CVE foi lançado em 1999 pela MITRE Corporation para identificar e catalogar vulnerabilidades em software em um conjunto de dados de livre acesso, para que organizações comerciais e governamentais possam melhorar sua segurança geral[15].

- **CWE:** é uma classificação e categorização de tipos comuns de vulnerabilidades de software. O objetivo do CWE é auxiliar na eliminação de vulnerabilidades identificando os erros mais comuns cometidos por desenvolvedores e engenheiros, para que eles evitem esses problemas nos produtos e sistemas que eles constroem. O CWE descreve fraque-

[13] <https://developer.datalogic.com/mobile-computers/news/android-rfp>
[14] <https://www.codiga.io/blog/cve-vs-cwe/>
[15] <https://www.bugcrowd.com/glossary/common-vulnerability-exposure-cve/>;<https://www.cve.org>

zas com uma taxonomia facilmente navegável usando linguagem comum, ajudando os desenvolvedores a verificar fraquezas em softwares e produtos existentes[16].

3.2.3 Erros de Usuário e Desenvolvedores

Erros cometidos tanto por usuários quanto por desenvolvedores contribuem significativamente para a ocorrência de vulnerabilidades.

- **Erros de Usuário:** Incluem práticas inseguras, como o download de aplicativos de fontes não confiáveis e a concessão de permissões excessivas a aplicativos.
- **Erros de Desenvolvedor:** Englobam falhas na implementação de segurança, como a falta de validação de entrada, configuração inadequada de permissões e uso de bibliotecas inseguras.

3.2.4 Métodos de Detecção de Vulnerabilidades

Os métodos de detecção de vulnerabilidades em sistemas operacionais móveis podem ser amplamente categorizados em três tipos[17]: análise estática, análise dinâmica e análise híbrida.

- **Análise Estática:** Esse método envolve examinar o código de um aplicativo sem executá-lo. A análise estática permite inspecionar o código-fonte ou, em alguns casos, o código executável (como binários ou bytecodes), para identificar padrões suspeitos, vulnerabilidades conhecidas (como uso inseguro de funções), e outras características potencialmente perigosas. Essa técnica é útil porque pode ser realizada rapidamente e não depende do ambiente de execução. **Ferramentas Comuns:** SAST (*Static Application Security Testing*) tools, como SonarQube e Fortify.
- **Análise Dinâmica:** Contrasta com a análise estática por executar o software em um ambiente controlado, a fim de observar seu comportamento em tempo real. Isso inclui monitorar as operações de sistema que o software executa, como chamadas de sistema, modificações de arquivos, e comunicações de rede. A análise dinâmica é valiosa porque

[16] <https://www.bugcrowd.com/glossary/common-weakness-enumeration-cwe>;<https://cwe.mitre.org>
[17] <https://www.ibm.com/think/topics/application-security>;<https://vfunction.com/blog/static-vs-dynamic-code-analysis/>

revela o comportamento do software sob condições operacionais reais, detectando atividades maliciosas que só ocorrem durante a execução. **Ferramentas Comuns:** DAST (*Dynamic Application Security Testing*) tools, como Burp Suite e ZAP.

- **Análise Híbrida:** Combina elementos das análises estática e dinâmica para formar uma visão mais completa do comportamento do software e suas potenciais vulnerabilidades. A análise híbrida pode correlacionar dados estáticos com comportamentos observados durante a execução, permitindo uma detecção mais precisa de técnicas evasivas ou complexas que poderiam ser perdidas se apenas um método de análise fosse usado. **Ferramentas Comuns:** Hybrid analysis platforms, que integram SAST e DAST.

3.2.5 Desafios na Detecção de Vulnerabilidades

A detecção de vulnerabilidades em aplicativos Android é uma tarefa complexa que envolve diversos desafios, tanto técnicos quanto operacionais. A seguir, são apresentados alguns dos principais obstáculos enfrentados por pesquisadores e desenvolvedores de segurança ao tentarem identificar e mitigar vulnerabilidades em sistemas Android.

- **Complexidade do Ecossistema Android:** O ecossistema Android é altamente fragmentado, com muita variedade de versões do SO em uso em diferentes dispositivos, além de customizações feitas por fabricantes e operadoras (ENCK et al., 2011). Essa fragmentação gera desafios na detecção de vulnerabilidades, pois o comportamento de um aplicativo pode variar significativamente entre diferentes dispositivos e versões do Android. Por exemplo, uma vulnerabilidade que afeta uma versão específica do Android pode não existir em outras versões ou dispositivos, tornando difícil garantir uma análise de segurança abrangente.

- **Evolução Constante de Aplicações e Bibliotecas:** Os aplicativos Android estão em constante desenvolvimento e atualização. Bibliotecas de código de terceiros são frequentemente integradas aos aplicativos para adicionar funcionalidades, e essas bibliotecas podem conter vulnerabilidades que passam despercebidas aos desenvolvedores do SO. Isto é, desenvolvedores podem não estar cientes de falhas nas bibliotecas que utilizam. Esse cenário dificulta a detecção de vulnerabilidades, especialmente quando bi-

bliotecas externas, particularmente as desatualizadas, introduzem novas vulnerabilidades nos aplicativos.

- **Falsos Positivos e Falsos Negativos:** As ferramentas de análise estática e dinâmica de código muitas vezes geram falsos positivos (identificação incorreta de vulnerabilidades) e falsos negativos (não identificação de vulnerabilidades existentes). Isso reduz a confiabilidade dessas ferramentas, gerando uma carga adicional de trabalho para a equipe de segurança, que precisa revisar manualmente os resultados.

  - Exemplo de falso positivo: Uma análise estática pode identificar como vulnerável um trecho de código que, na prática, não é explorável devido a outros mecanismos de segurança. A falha foi detectada, porém está protegida por mecanismos de segurança como firewalls, sistemas e autenticação, criptografia, não sendo passível de ataques.
  - Exemplo de Falso Negativo: Uma vulnerabilidade pode não ser detectada em uma análise dinâmica se o código vulnerável não for executado durante os testes.

    Vale ressaltar que a ocorrência de falsos positivos e falsos negativos não decorre apenas de limitações das técnicas empregadas ou de falhas no sistema em teste, mas também pode estar associada a erros na execução dos próprios testes, o que compromete a confiabilidade dos resultados.

- **Execução de Código Dinâmico e Ofuscação:** Muitos desenvolvedores usam técnicas de ofuscação para proteger seus aplicativos e impedir a engenharia reversa. No entanto, essas técnicas também dificultam a análise automatizada, tornando mais difícil para ferramentas de segurança identificar vulnerabilidades dentro do código. A ofuscação de código impede que as ferramentas de análise estática identifiquem vulnerabilidades com precisão, forçando os pesquisadores a usar técnicas avançadas de engenharia reversa ou monitoramento dinâmico.

- **Integração no Ciclo de Desenvolvimento de Software (SDLC):** A detecção de vulnerabilidades deve ser integrada de forma eficaz no ciclo de vida do desenvolvimento de software (SDLC). Isso inclui a detecção durante o desenvolvimento, na fase de testes e no pós-lançamento. Em muitos casos, a detecção de vulnerabilidades ocorre tardiamente, quando o aplicativo já foi lançado, o que pode comprometer a segurança dos usuários.

Isso traz um *desafio operacional*. Em ciclos de desenvolvimento ágeis ou DevOps, a velocidade com que novos recursos e atualizações são implementados pode dificultar a introdução de práticas adequadas de segurança. Isso faz com que vulnerabilidades não sejam detectadas antes do lançamento.

- **Obsolescência de Ferramentas de Segurança:** As ferramentas de segurança precisam acompanhar a evolução das técnicas de ataque. Ferramentas de análise estática e dinâmica, quando desatualizadas, muitas vezes não conseguem identificar novas vulnerabilidades, deixando desenvolvedores e empresas vulneráveis a formas de exploração que não existiam ou não eram conhecidas nas versões anteriores dessas ferramentas (ENCK et al., 2011).

- **Restrições de Performance e Recursos Móveis:** A detecção de vulnerabilidades em dispositivos móveis é limitada pela capacidade de processamento, memória e consumo de energia dos dispositivos. Ao contrário dos servidores ou desktops, onde análises completas podem ser realizadas sem grandes restrições, os dispositivos móveis não suportam análises intensivas em termos de recursos sem afetar a experiência do usuário. Por exemplo, executar uma análise dinâmica intensiva pode causar lentidão perceptível no dispositivo móvel, o que é impraticável em ambientes de uso real, onde os recursos limitados não suportam tarefas pesadas sem degradação de desempenho.

- **Privacidade e Dados Sensíveis:** A análise de vulnerabilidades em aplicativos Android muitas vezes requer o monitoramento de dados sensíveis transmitidos pelos aplicativos. No entanto, garantir a privacidade desses dados durante os testes pode ser um desafio, uma vez que informações sensíveis podem ser expostas ou usadas de forma inadequada durante o processo de detecção de vulnerabilidades. Isso traz um *desafio ético*. Ferramentas de análise precisam ser projetadas para garantir que os dados sensíveis dos usuários sejam protegidos durante a detecção de vulnerabilidades, especialmente ao lidar com aplicativos que acessam informações pessoais, como dados bancários ou de saúde.

Como visto acima, a detecção de vulnerabilidades em aplicativos Android enfrenta uma série de desafios técnicos, operacionais e éticos. A complexidade do ecossistema Android, a evolução constante das técnicas de ataque, os falsos positivos e falsos negativos, a ofuscação de código e a integração ineficaz no SDLC são alguns dos principais obstáculos que tornam a identificação e a mitigação de vulnerabilidades uma tarefa árdua (SENANAYAKE et al., 2023).

Para superar esses desafios, é essencial a adoção de ferramentas de detecção atualizadas, técnicas avançadas de análise, e a implementação de práticas de segurança desde o início do ciclo de desenvolvimento.

3.3 RELATOS DE VULNERABILIDADES NA INTERNET

A internet está repleta de relatos de vulnerabilidades em dispositivos e sistemas, sendo encontrados em forma de texto e vídeos no YouTube. Esses relatos são de importância central para a rápida identificação e correção de falhas em sistemas operacionais e aplicativos, complementando assim as outras abordagens de identificação de vulnerabilidades via teste de software.

Como mencionado acima, os relatos podem se apresentar em diferentes formatos, dependendo da plataforma e do público-alvo. Destacam-se aqui os textos em fóruns e blogs técnicos, e os vídeos disponíveis no YouTube.

Fóruns como **XDA Developers**[18] e **Stack Overflow**[19] frequentemente recebem relatos de vulnerabilidades de usuários e de especialistas. Esses relatos geralmente vêm acompanhados de instruções detalhadas sobre como explorar falhas em SOs ou aplicativos. Um relato típico pode descrever uma vulnerabilidade no gerenciamento de permissões de um aplicativo Android, com um passo a passo de como modificar o sistema para ganhar acesso não autorizado a dados sensíveis.

Atualmente, o **YouTube** tornou-se uma plataforma muito utilizada para a divulgação de relatos de vulnerabilidades, onde usuários disponibilizam vídeos com tutoriais mostrando como explorar uma falha de segurança em tempo real. Por exemplo, um vídeo pode mostrar como desativar a *Proteção de Reset de Fábrica* (do inglês, *Factory Reset Protection - FRP*) em dispositivos Android, fornecendo uma demonstração visual do processo.

Vale salientar que esses relatos de vulnerabilidades são postados em diversos idiomas, incluindo inglês, português, espanhol e muitos outros. O idioma do conteúdo (artigo) publicado depende da base de usuários da plataforma. Por exemplo, em fóruns brasileiros, relatos geralmente são postados em português, contendo gírias e jargões técnicos locais, o que dificulta o entendimento dos relatos por desenvolvedores que não são fluentes na língua.

[18] <https://www.xda-developers.com/>
[19] <https://stackoverflow.com/>

3.3.1 Desafios na Análise de Relatos

Tratar automaticamente esses relatos de vulnerabilidades apresenta desafios significativos. Alguns dos principais problemas estão comentados a seguir. O Capítulo 4 irá apresentar exemplos de tipos de relatos, destacando as dificuldades na sua identificação, obtenção e interpretação automática.

- **Variedade de formatos e estrutura:** Relatos podem ser apresentados em textos longos e detalhados ou breves e fragmentados. Além disso, vídeos com explicações verbais dificultam a extração automática de informações relevantes, sendo importante realizar-se a transcrição das falas.

- **Irregularidades e uso de gírias:** Muitos relatos são escritos de maneira informal, com o uso de gírias, abreviações e palavras truncadas, o que torna difícil a análise automatizada usando métodos tradicionais de Processamento de Linguagem Natural.

- **Linguagem técnica e jargões:** Esses relatos frequentemente utilizam jargões técnicos que variam de acordo com o tipo de vulnerabilidade. Ferramentas automáticas têm dificuldade em compreender e categorizar esse tipo de conteúdo sem um treinamento especializado em cada domínio técnico.

Como já mencionado, esses relatos são essenciais para a identificação de novas vulnerabilidades, antes que elas sejam exploradas em larga escala. Em geral, esses relatos detalham cada passo da sua execução. Assim, os desenvolvedores e pesquisadores de segurança utilizam essas informações para realizar e lançar correções (*patches*) de maneira proativa, evitando que os invasores possam explorar as falhas descobertas.

Mesmo com os desafios mencionados, o monitoramento constante dessas fontes de informação é vital. Relatos de vulnerabilidades fornecem informações ricas que nem sempre estão disponíveis em outras fontes mais formais, como bancos de dados de vulnerabilidades (ex: *Common Vulnerabilities and Exposures* - CVE[20]), e podem ser a primeira linha de defesa contra novos tipos de ataques.

[20] <https://cve.mitre.org/>

3.4 TRABALHOS RELACIONADOS À IDENTIFICAÇÃO DE VULNERABILIDADES

Esta seção trata de trabalhos que, assim como a pesquisa aqui relatada, objetivam identificar vulnerabilidades no SO Android. Como não identificamos na literatura disponível trabalhos de detecção de vulnerabilidades baseados em análise de texto, restringimos esta seção a apresentar trabalhos que se baseiam nas abordagens tradicionais de teste de software. Essas abordagens foram brevemente apresentadas na Seção 3.2.4.

3.4.1 Análise Estática

A análise estática em Android é amplamente utilizada para detectar vulnerabilidades sem a necessidade de executar o código. Essa abordagem permite inspecionar o código-fonte ou executável em busca de padrões suspeitos e vulnerabilidades conhecidas, sendo amplamente utilizada em ferramentas de teste de segurança.

No estudo conduzido por (BAYAZIT; SAHINGOZ; DOGAN, 2022), os autores desenvolveram um modelo de detecção de malware baseado em aprendizado profundo que utiliza a análise estática para inspecionar aplicativos Android. O modelo extrai características diretamente dos arquivos APK, como permissões e intents, e as processa com técnicas avançadas de aprendizado profundo, incluindo RNN, LSTM, BiLSTM e GRU.

Para avaliar o desempenho do modelo, os autores utilizaram o conjunto de dados CICInvesAndMal2019[21], que contém amostras de aplicativos benignos e maliciosos. Os resultados mostraram que o modelo BiLSTM obteve uma taxa de precisão de 98,85%, superando os demais métodos testados, além de manter uma baixa taxa de falsos positivos. Esses resultados indicam que o sistema é eficiente e confiável na classificação de malware.

Já o trabalho de (KARIM; CHANG; FIRDAUS, 2020) explora padrões estáticos como grafos de chamadas de função e uso de permissões para detectar comportamento malicioso, mostrando-se uma técnica eficaz e econômica para detecção de malware em grande escala. Essa técnica é especialmente útil para ambientes com recursos limitados, como dispositivos móveis.

[21] <https://www.kaggle.com/datasets/malikbaqi12/cic-invesandmal2019-dataset>

3.4.2 Análise Dinâmica

A análise dinâmica é essencial para a detecção de vulnerabilidades que se manifestam durante a execução de aplicativos em ambientes de teste realistas, como sandboxes. Diferente da análise estática, que examina o código sem executá-lo, a análise dinâmica permite observar o comportamento do aplicativo em tempo real, revelando atividades maliciosas e falhas que ocorrem apenas durante o uso.

No estudo de (GHORBANI et al., 2023), os autores apresentam o DeltaDroid, uma ferramenta projetada especificamente para testes de entrega dinâmica em aplicativos Android. O DeltaDroid realiza testes com os *Dynamic Feature Modules* (DFMs), que são módulos de funcionalidades instalados sob demanda após a instalação inicial do aplicativo. Essa ferramenta permite a simulação e o monitoramento da instalação dos DFMs em condições reais de uso, como interrupções de rede e falta de espaço de armazenamento.

A principal contribuição do DeltaDroid reside na sua capacidade de expandir a cobertura de testes em um ambiente que replica situações adversas, algo que os métodos de análise estática e ambientes de desenvolvimento convencionais não conseguem simular com precisão. Ao permitir que desenvolvedores testem a instalação dos DFMs em tempo real, o DeltaDroid detecta vulnerabilidades e problemas de desempenho ocultos, oferecendo uma análise mais robusta e adaptada a cenários reais. Os resultados do estudo foram positivos: o DeltaDroid obteve uma taxa de precisão superior a 90% na detecção de falhas específicas de instalação em DFMs, revelando problemas que outros métodos de teste convencionais não capturam. Com sua alta taxa de detecção e a capacidade de gerar casos de teste específicos para diferentes cenários de falha, o DeltaDroid é uma ferramenta essencial para desenvolvedores que buscam uma cobertura mais completa e precisa dos testes de aplicativos Android.

3.4.3 Análise Híbrida

A análise híbrida une as vantagens das análises estática e dinâmica, permitindo uma detecção de vulnerabilidades mais abrangente e eficiente. Essa abordagem é capaz de detectar tanto vulnerabilidades conhecidas, como padrões de código malicioso e ainda comportamentos suspeitos em tempo de execução.

No contexto do Android, métodos híbridos são particularmente úteis para enfrentar os desafios de fragmentação e customização do sistema operacional, uma vez que combinam

inspeções detalhadas de código com observações de comportamento dinâmico. Os trabalhos de (CHEN et al., 2018) e (KARIM; CHANG; FIRDAUS, 2020) foram selecionados para uma breve discussão.

(CHEN et al., 2018) destacam que, embora a análise dinâmica seja mais robusta contra técnicas de evasão, como a ofuscação, ela exige infraestrutura adicional e enfrenta dificuldades com cobertura de código, ou seja, nem todos os caminhos do programa são executados e testados. Esse desafio leva à necessidade de uma análise híbrida que combine os pontos fortes das análises estática e dinâmica para uma cobertura mais completa.

Nesse contexto, (CHEN et al., 2018) desenvolveram o TinyDroid, mostrando que a análise híbrida pode melhorar a taxa de detecção e reduzir os falsos positivos ao integrar técnicas de simplificação de opcode (*operation code*) com análise de frequência de N-grams e aprendizagem de máquina. Isso permite que o sistema detecte não apenas malwares conhecidos, mas também variantes desconhecidas, o que é essencial em um ambiente de ameaças em rápida evolução, como o Android. O TinyDroid mostrou resultados bastante positivos na detecção de malwares Android, superando várias ferramentas antivírus tradicionais em ambientes de teste. Em experimentos com um conjunto de dados extenso (baseado no Drebin Dataset), o TinyDroid obteve uma taxa de detecção de malware de 98,6% em um conjunto de 4000 amostras, demonstrando sua eficácia em identificar até mesmo variantes desconhecidas de malware. A taxa de falso-positivos foi mantida em 1,4%, destacando o modelo como preciso e confiável na classificação de amostras benignas e maliciosas.

Já o trabalho de (KARIM; CHANG; FIRDAUS, 2020) apresenta uma estrutura híbrida para detectar botnets[22] móveis, utilizando análise estática para capturar permissões e grafos de chamadas, e análise dinâmica para observar o comportamento em tempo de execução, como envio de mensagens e controle remoto de dispositivos. Com uma taxa de precisão de até 98%, o método desenvolvido se destaca pela alta taxa de detecção e por minimizar falsos positivos, proporcionando uma solução eficiente para a segurança móvel em um cenário de ameaças em rápida evolução. Além disso, os testes demonstraram que a estrutura híbrida é eficaz para identificar atividades de controle remoto e comunicação entre dispositivos infectados, um comportamento típico de botnets.

[22] Botnet – rede de computadores infectados por malware e controlados remotamente por hackers.

3.5 CONSIDERAÇÕES FINAIS

Como visto, as inúmeras possibilidades de ataques a dispositivos móveis motivam estudos sérios na tentativa de identificar e mitigar tais perigos, tendo em vista a grande importância desses dispositivos na vida atual.

Os trabalhos desenvolvidos com o objetivo de tratar vulnerabilidades ainda utilizam as abordagens predominantes da área de teste de software: análise estática de código, análise dinâmica e análise híbrida. As maiores novidades nesses trabalhos estão relacionadas ao uso de técnicas de aprendizagem de máquina, como uma colaboração entre as áreas de pesquisas de Engenharia de Software e Inteligência Artificial.

Contudo, não encontramos trabalhos com foco na busca e utilização de relatos na Web sobre vulnerabilidades, o que nos ofereceu uma lacuna a ser explorada.

O Capítulo 4, a seguir, descreve o processo proposto para identificar relatos na Web e extrair deles informações relevantes para tratar esses problemas. Já o Capítulo 5 traz detalhes da implementação do protótipo e resultados de testes realizados.

4 IDENTIFICAÇÃO E EXTRAÇÃO DE INFORMAÇÃO DE VULNERABILIDADES DO ANDROID

Este capítulo descreve o trabalho que foi o principal foco desta pesquisa de mestrado: um software para auxiliar na detecção de vulnerabilidades de SO Android. A Seção 4.1 apresenta o contexto do trabalho, ressaltando os fatores que motivaram seu desenvolvimento, bem como sua originalidade no cenário de dispositivos móveis. A Seção 4.1.1 trata de detecção de vulnerabilidades no Android, e a Seção 4.1.2 descreve o processo manual de identificação de vulnerabilidades que era realizado pela empresa parceira antes do desenvolvimento da solução proposta aqui. A Seção 4.1.3 apresenta a solução proposta, dando uma visão geral do processo de detecção implementado.

A seguir, as seções 4.2 a 4.6 detalham os módulos e arquivos auxiliares do sistema implementado. Por fim, a Seção 4.7 traz as considerações finais deste capítulo.

4.1 CONTEXTO E TRABALHO REALIZADO

Como discutido anteriormente (Capítulo 2), os testes de software são uma etapa essencial do Ciclo de Vida do Desenvolvimento de Software, sendo indispensáveis para garantir a qualidade e a segurança de qualquer produto tecnológico (SOMMERVILLE, 2015). Embora existam diferentes tipos e estratégia de testes, todos compartilham o mesmo objetivo: identificar e corrigir falhas, bugs e erros que possam comprometer o funcionamento ou a segurança dos sistemas.

Especificamente, este trabalho teve como objetivo geral realizar um estudo na área de identificação de vulnerabilidades em sistemas operacionais para dispositivos móveis (Capítulo 3), que se tornaram indispensável no nosso dia a dia. A detecção de vulnerabilidades nos SOs desses dispositivos é um aspecto crucial na segurança digital, pois falhas no sistema podem comprometer a privacidade dos usuários e permitir a exploração indevida dos dispositivos.

Este trabalho foi conduzido no contexto de uma colaboração na área de Teste de SW entre o CIn-UFPE e a Motorola Mobility, uma empresa global especializada no desenvolvimento de dispositivos móveis. Escolhemos investigar a identificação de vulnerabilidades no SO Android, que é utilizado nos dispositivos móveis dessa empresa. A *relevância* dessa escolha pode ser justificada pelo fato de que o SO Android é utilizado na maioria dos dispositivos móveis

atualmente no mercado[1].

Contudo, como será visto neste texto, a solução proposta também pode ser usada para buscar vulnerabilidades em outros produtos e outros SOs, bastando para isto um ajuste de parâmetros da coleta (ver Seção 4.2).

Outra questão importante a ressaltar é a originalidade da pesquisa desenvolvida. A Motorola Mobility adota processos rigorosos de teste e segurança para garantir a qualidade de seus produtos, monitorando continuamente relatos de vulnerabilidades de diversas fontes, como testes internos, fóruns de usuários, redes sociais e pesquisas independentes. Contudo, devido à grande quantidade de informações dispersas na Internet, o monitoramento manual para detecção de vulnerabilidades torna-se ineficiente e demorado.

Assim, esta pesquisa buscou oferecer um processo semiautomatizado, utilizando Web Scraping e Mineração de Texto para a coleta e análise de dados relevantes com precisão e rapidez. Com essa solução, a Motorola poderá reduzir o tempo de resposta na correção de falhas, eliminar a necessidade de buscas manuais e aumentar a precisão na detecção de vulnerabilidades, fortalecendo a segurança de seus dispositivos.

Como já mencionado, acreditamos que este é um tema original, uma vez que não encontramos na literatura relacionada nenhum trabalho com foco na detecção automática de vulnerabilidades através da análise de relatos dispersos na internet.

4.1.1 Detecção de vulnerabilidades no Android

Para mitigar os riscos, a Motorola adota uma abordagem abrangente de testes, combinando estratégias manuais e automatizadas para identificar e corrigir vulnerabilidades antes que possam ser exploradas por atacantes.

O processo de detecção ocorre em várias etapas, que podem ser descritas como abaixo:

- Testes Internos Pré-Lançamento – Antes de uma nova versão do Android Motorola ser disponibilizada, ela passa por uma série de testes internos, incluindo inspeção de código-fonte, execução de testes de segurança automatizados e simulação de ataques.

- Testes de Integração – O sistema Android básico desenvolvido pelo Google recebe adaptações específicas da Motorola. Durante essa fase, novos recursos são implementados e testados para garantir compatibilidade e segurança.

[1] <https://canaltech.com.br/software/qual-o-sistema-operacional-de-celular-mais-usado-do-mundo-223862>

- Testes em Ambiente Controlado – São realizados testes em dispositivos físicos e em simuladores, avaliando o comportamento do sistema em diferentes cenários de uso. Essa fase busca detectar falhas relacionadas a permissões, criptografia e execução de código não autorizado.

- Monitoramento Pós-Lançamento – Após a distribuição da nova versão do Android Motorola, a empresa continua monitorando relatos de falhas reportadas por usuários, pesquisadores e especialistas em segurança.

Apesar da eficácia desse processo, vulnerabilidades ainda podem passar despercebidas, sendo detectadas apenas na etapa de pós-lançamento. Muitos relatos de vulnerabilidades surgem em discussões online, algumas sendo descobertas por hackers antes mesmo das equipes de segurança. Tais relatos aparecem em fóruns como XDA Developers, redes sociais e vídeos no YouTube, onde usuários compartilham métodos para contornar proteções do Android, incluindo a *Factory Reset Protection* (FRP).

Esses relatos têm sido monitorados manualmente, seguindo um processo descrito na seção a seguir. Contudo, fica claro que, devido à grande quantidade de relatos espalhados na Internet, esse processo manual é custoso e algumas vezes ineficaz.

4.1.2 Cenário empresarial - Processo manual de detecção de vulnerabilidades

Antes da implementação do sistema automatizado de monitoramento, a detecção de vulnerabilidades baseada em relatos online era realizada manualmente pela equipe de segurança da Motorola. Esse processo envolvia buscas ativas em fóruns, redes sociais e sites especializados para identificar possíveis falhas relatadas por usuários.

O fluxo de trabalho manual inclui as seguintes etapas:

- Busca Manual por Relatos – Os especialistas utilizavam motores de busca (como Google) para navegar por fóruns técnicos, canais no YouTube e sites especializados. As fontes mais consultadas incluíam XDA Developers, Stack Overflow, BypassFRPFiles e canais de técnicos independentes em segurança móvel.

- Filtragem e Análise dos Relatos – Após a coleta das informações, os especialistas avaliam a relevância e a confiabilidade dos relatos, verificando se os métodos descritos representam falhas reais de segurança.

- Registro e Classificação – As informações consideradas relevantes eram documentadas manualmente, associadas a dispositivos específicos e registradas para posterior investigação.
- Encaminhamento para Investigação – Os relatos validados eram então encaminhados às equipes de desenvolvimento e qualidade para posterior verificação técnica e, se necessário, correção.

Esse processo apresenta diversos desafios:

- Alto Volume de Informações – Com o crescente número e variedade de relatos publicados diariamente, a busca manual visando uma alta cobertura na identificação de novos relatos se tornou impraticável.
- Tempo Elevado de Resposta – A análise manual demanda muito tempo, atrasando a detecção e correção das falhas.
- Duplicidade de Relatos – Existe um número considerável de relatos distintos na web que tratam da mesma vulnerabilidade, gerando redundâncias no trabalho realizado pela equipe de segurança.
- Dificuldade na Verificação de Relevância – Nem todas as informações coletadas são realmente sobre vulnerabilidades ou aplicáveis aos dispositivos da Motorola.
- Dependência de análises individuais – O processo manual dependia fortemente da interpretação individual, o que resultava em inconsistências na detecção de vulnerabilidades.

Diante dessas dificuldades, tornou-se evidente a necessidade de automatizar a identificação e análise de vulnerabilidades reportadas na web. A solução proposta e implementada será detalhada nas seções a seguir.

4.1.3 Solução proposta

Com o objetivo de superar as limitações do processo manual de detecção de vulnerabilidades descrito na seção anterior, este trabalho propôs e desenvolveu uma solução semi-automatizada baseada em técnicas de Mineração de Texto, Web Scraping e Processamento de Linguagem

Natural (PLN). A proposta visa acelerar a identificação de relatos relevantes sobre vulnerabilidades em dispositivos Android, especialmente aqueles associados ao mecanismo de proteção *Factory Reset Protection* (FRP).

A solução desenvolvida integra múltiplas etapas automatizadas que vão desde a coleta de dados em fontes públicas da Web até a exibição estruturada e ranqueada dos relatos por meio de uma interface interativa. Essas etapas foram desenhadas para minimizar o esforço manual da equipe de segurança e garantir maior agilidade na identificação de falhas emergentes. A Figura 5 traz uma visão geral do processo proposto, apresentando o fluxo de dados e processos do sistema implementado.

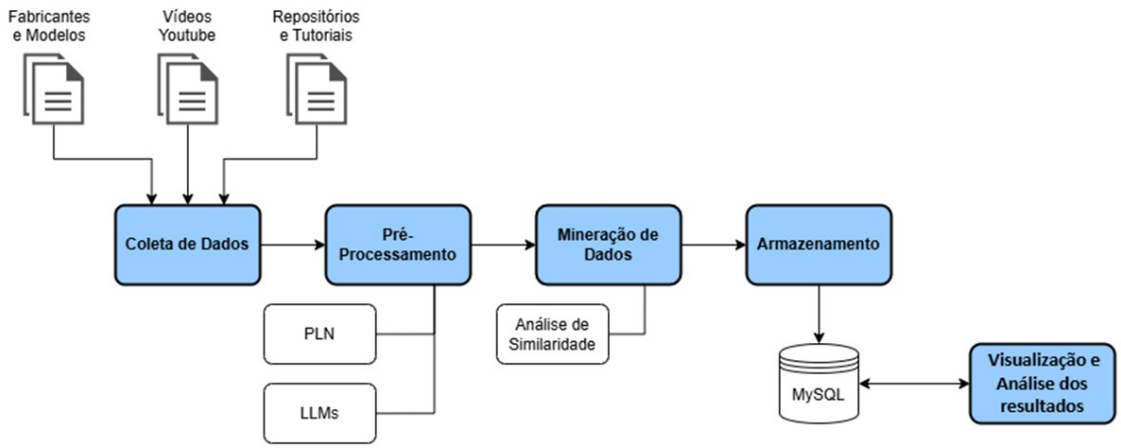

Figura 5 – Fluxo - Visão Geral do Sistema. Fonte: Autor

A seguir, a Seção 4.1.3.1 destaca os objetivos do trabalho realizado. O detalhamento dos documentos de entrada e dos módulos de processamento serão vistos em seções específicas a seguir.

4.1.3.1 Objetivos da Solução Proposta

Os principais objetivos da solução proposta são:

- Aprimorar a segurança de dispositivos Android, permitindo a detecção e correção ágil de

vulnerabilidades logo após sua divulgação pública, reduzindo o risco de novas explorações por agentes maliciosos.

- Automatizar a detecção de vulnerabilidades reportadas na Web, eliminando a necessidade de buscas manuais e otimizando o processo de análise.
- Ampliar a cobertura da análise e identificar vulnerabilidades que poderiam passar despercebidas em buscas convencionais.
- Estruturar e categorizar de forma precisa relatos informais. Aplicando técnicas avançadas de Processamento de Linguagem Natural (PLN) e Modelos de IA.
- Implementar um banco de dados centralizado, garantindo o armazenamento eficiente e a organização dos relatos coletados para consultas e análises futuras.
- Fornecer uma interface intuitiva, que permita a visualização clara e acessível dos dados extraídos, possibilitando análises rápidas e fundamentadas por especialistas.

As seções a seguir apresentam detalhes sobre os documentos de entrada e como se dá sua coleta (Seção 4.2), seguida dos outros módulos do protótipo implementado (Seção 4.3 em diante). Por fim, a Seção 4.7 traz as considerações finais deste capítulo.

4.2 COLETA DE DADOS

A coleta de dados representa a etapa inicial do processo automatizado de identificação de vulnerabilidades. Nesta fase, o sistema é responsável por buscar e extrair informações relevantes sobre falhas de segurança em dispositivos Android, a partir de diferentes fontes da web. Esta seção apresenta brevemente os tipos de documentos atualmente coletados pelo sistema desenvolvido, sendo todos em formato textual. Como mostra a Figura 5 (Visão Geral do Sistema), dados são coletados principalmente a partir do site BypassFRPFiles, que reúne tutoriais técnicos e arquivos relacionados ao desbloqueio de dispositivos Android, bem como a partir de transcrições de relatos extraídos de vídeos do YouTube.

Inicialmente, o módulo coleta informações atualizadas sobre fabricantes e modelos de dispositivos (Seção 4.2.1). A seguir, com base nas informações sobre fabricantes e modelos, o sistema realiza a coleta de informações sobre vulnerabilidades encontradas em sites específicos (Seção 4.2.2) e a partir de vídeos no YouTube (Seção 4.2.3).

Como já mencionado, o foco inicial dessa coleta foi voltado para o *Factory Reset Protection* (FRP) Bypass, por ser uma vulnerabilidade que traz grandes riscos à segurança desses dispositivos. Contudo, como ficará claro ao longo do texto, o sistema pode ser facilmente estendido para lidar com outros tipos de vulnerabilidades.

4.2.1 Fabricantes, Modelos e Versões do Android

Antes de iniciar a coleta específica de relatos de vulnerabilidade, é necessário identificar a lista de fabricantes e modelos de dispositivos disponíveis no mercado, bem como suas respectivas versões do sistema Android. Essa informação serve como base para filtrar os relatos coletados, garantindo que apenas aqueles relevantes à Motorola (ou ao escopo definido) sejam considerados.

Para isso, foi desenvolvido um módulo de Web Scraping no site GSM Arena[2], uma das maiores bases públicas de dados sobre smartphones (ver Figura 6). Esse módulo percorre páginas HTML e extrai de forma automatizada os seguintes elementos:

- Nome do fabricante (e.g., Motorola, Samsung)
- Modelo do dispositivo (e.g., Moto G100, Moto G60)
- Versões compatíveis do Android (e.g., Android 11, Android 12)

Foi criado um script automático para realizar essa coleta (ver Capítulo 5, seções 5.1.2 e 5.1.4.1).

As informações coletadas são organizadas em tabelas relacionais no banco de dados MySQL, permitindo sua indexação e uso nos módulos posteriores de coleta e análise.

O processo automático de extração da informação utilizada para preencher essa tabela (figura 7) teve 100% de acerto, uma vez que os dados estão rigidamente formatados no site de origem. Existe apenas uma situação em que ocorrem erros, porém trata-se de um bloqueio de segurança do site. Isso ocorre quando tentamos realizar vários acesso consecutivos.

[2] <https://www.gsmarena.com/>

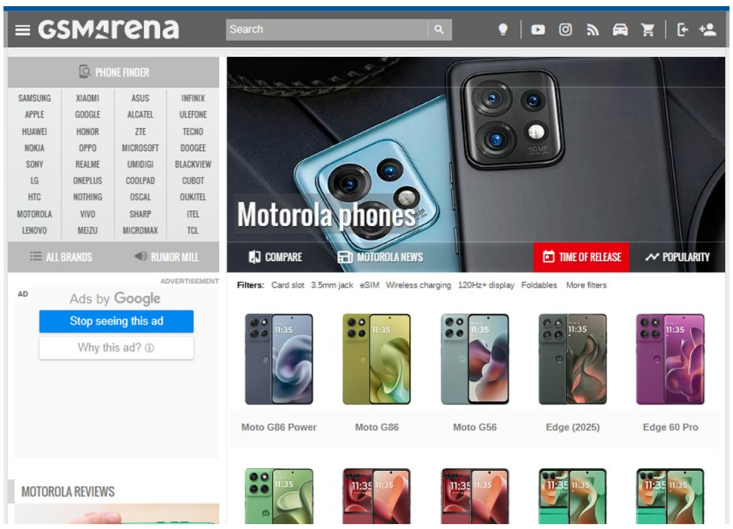

Figura 6 – Exemplo de página com dados sobre Smartphones. Fonte: Site GSM Arena

model_names	vendor_name	android_versions
Moto E7 Plus	Motorola	-
Moto E20	Motorola	11
Moto G04	Motorola	14
Moto G9 Play	Motorola	11
Moto G100	Motorola	12
Moto E40	Motorola	11
Moto G22,Moto G60,Moto G60S	Motorola	12,13
Moto E6i,Moto E6s	Motorola	9
One Fusion	Motorola	11
Moto E7,Moto E7 Plus,Moto E7 Power	Motorola	-
Moto E13	Motorola	13
Moto G8 Power Lite	Motorola	10,11

Figura 7 – Trechos da tabela do BD armazena fabricante, modelo e versão do Android

4.2.2 Repositórios Específicos

Com base nas informações previamente coletadas sobre fabricantes, modelos e versões do Android, o sistema inicia a busca automatizada por relatos de vulnerabilidades em fontes especializadas. Diversos sites e bases de dados online disponibilizam informações relevantes. Podemos citar, por exemplo, os boletins de segurança do Android (disponíveis no site do Android Open Source project[3]), o banco de dados CVE Details[4], o repositório de vulnerabilidades da MITRE (CVE/Mitre)[5], além de fóruns técnicos como o Reddit[6] e XDA Developers[7], nos

[3] <https://source.android.com/docs/security/bulletin/2025-03-01?hl=pt-br>
[4] <https://www.cvedetails.com/cve/CVE-2020-15580/>
[5] <https://www.cve.org/CVERecord?id=CVE-2024-40650>
[6] <https://www.reddit.com/r/SonyXperia/comments/1cyvnrt/how_do_i_disable_or_remove_frp_i_cant_for_the/>
[7] <https://xdaforums.com/t/guide-citrus-how-to-remove-frp-lock.4491833/>

quais usuários frequentemente compartilham descobertas não documentadas oficiais.

A padronização, ainda que parcial, presente nessas fontes facilita a adaptação do módulo de coleta. Em geral, os relatos seguem algum padrão de formatação, o que permite a criação de scripts de Web Scraping para extrair automaticamente os dados de interesse.

Neste trabalho, o foco inicial foi voltado para vulnerabilidades relacionadas ao *Factory Reset Protection* (FRP) em dispositivos Android. A coleta automatizada foi realizada, especificamente, a partir do site BypassFRPFiles[8], que reúne arquivos, tutoriais e ferramentas voltadas para o desbloqueio de dispositivos móveis. Embora o site cubra diversas marcas, o sistema desenvolvido filtra apenas os relatos relacionados a dispositivos da Motorola, utilizando como critério os nomes de fabricantes e modelos previamente extraídos.

A coleta ocorre a partir da página principal, como mostrado na Figura 8, acessando os links de cada postagem por meio da opção "Read more". Com isso, o sistema extrai dados como títulos, descrições detalhadas dos métodos de desbloqueio e links para ferramentas técnicas associadas.

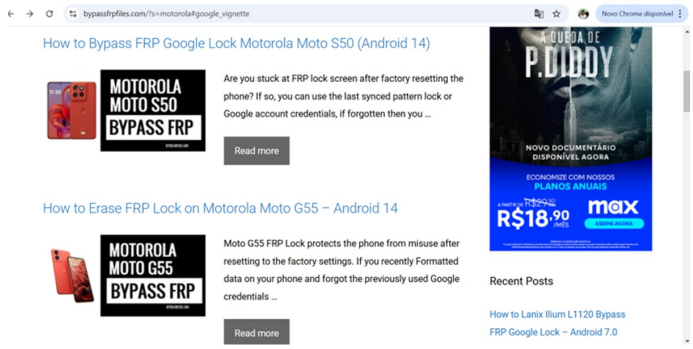

Figura 8 – Exemplo de relatos sobre FRP Bypass no Android Motorola. Fonte: <https://bypassfrpfiles.com/> - Acesso em 17/03/2025

A Figura 9 traz mais um exemplo de relato extraído do site FRP Bypass.

O código responsável por essa etapa foi desenvolvido com base em técnicas de Web Scraping e será descrito com mais detalhes no Capítulo 5, seções 5.1.2 e 5.1.4.1. A padronização das publicações nesse repositório facilitou a extração automática das informações, garantindo que os procedimentos coletados sejam organizados de forma estruturada no banco de dados.

Por fim, cabe ressaltar que a solução proposta é flexível: pode ser facilmente adaptada para coletar informações sobre outros fabricantes, sistemas operacionais e tipos de vulnerabilidades, bastando, para isso, ajustar os parâmetros de entrada definidos nos arquivos de configuração.

[8] <https://bypassfrpfiles.com/>

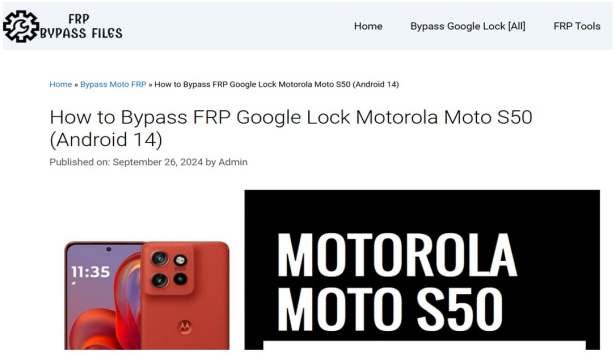

Figura 9 – Outro exemplo de relato sobre FRP Bypass. Fonte: <https://bypassfrpfiles.com/> - Acesso em 17/03/2025

id	title	publication_date	link
1	REMOVER CONTA GOOGLE MOTO G54 ANDROID 14 sem PC MÉTODO INFALÍVEL	2024-08-05 00:00:00	https://www.youtube.com/watch?v=RVLF
2	VALE ATÉ 2027 / REMOVER CONTA GOOGLE MOTOROLA G62 G42 G41 G52 G30 G31 G3...	2023-04-17 00:00:00	https://www.youtube.com/watch?v=7pRt
3	SAIUU !!! REMOVER CONTA GOOGLE MOTO G30 MOTO G31 ANDROID 12 SEM PC MÉTO...	2024-03-23 00:00:00	https://www.youtube.com/watch?v=R09K
4	DESBLOQUEIO CONTA GOOGLE MOTO G84 ANDROID 14 SEM PC MÉTODO EXCLUSIVO 1...	2024-07-31 00:00:00	https://www.youtube.com/watch?v=DB4Z
5	How to Bypass FRP Google Lock Motorola Edge 50 (Android 14)	2024-08-19 09:01:13	https://bypassfrpfiles.com/2024/08/frp-mo
6	Motorola Razr Plus 2024 Bypass FRP Google Verification Lock [Android 14]	2024-08-17 22:26:42	https://bypassfrpfiles.com/2024/08/motor
7	How to Motorola Edge (2024) Bypass FRP Google Lock [Android 14]	2024-08-17 21:41:15	https://bypassfrpfiles.com/2024/08/motor
8	Bypass Google FRP on Motorola Defy 2 [Android 12 Fix YouTube Update]	2024-08-04 22:58:41	https://bypassfrpfiles.com/2024/07/bypas
9	How to Bypass Google Gmail Verification FRP Lock on Motorola Moto G04s, G04 [Latest]	2024-08-04 15:39:55	https://bypassfrpfiles.com/2024/07/bypas
10	Bypass Google FRP on Motorola Moto G Play (2023) [Android 12 Fix YouTube Update]	2024-08-04 21:59:05	https://bypassfrpfiles.com/2024/07/bypas
11	How to Motorola G54 MDM Lock Remove [Step-by-Step] Full Guide	2024-07-20 09:11:16	https://bypassfrpfiles.com/2024/07/remov
12	How to Bypass Google FRP Lock on Motorola Moto G13 [Android 13 Fix YouTube Update]	2024-08-04 16:43:30	https://bypassfrpfiles.com/2024/07/bypas
13	Motorola Razr 50 Ultra Android 14 Bypass Google Verification FRP Lock [Latest]	2024-08-04 08:49:14	https://bypassfrpfiles.com/2024/07/motor
14	How to Bypass Google FRP Lock on Motorola Moto G72 [Fix YouTube Update]	2024-08-04 12:08:28	https://bypassfrpfiles.com/2024/07/bypas
15	remover conta Google moto e22 Android 12 sem pc método super novo sem pc	2023-06-06 00:00:00	https://www.youtube.com/watch?v=-91v
16	SUPER BOX ? GRATIS REMOVER CONTA GOOGLE FRP BYPASS DE TODOS OS MOTOROL...	2021-03-12 00:00:00	https://www.youtube.com/watch?v=Ryg-
17	desvincular conta google moto e7 / e7 play / e7 Power / e7 plus Android 10 ou 11 SEM PC	2023-01-01 00:00:00	https://www.youtube.com/watch?v=qyW

Figura 10 – Trecho no BD com resultados do scraping

Como no caso anterior, o processo automático de extração de informação desenvolvido teve 100% de acerto, uma vez que os dados extraídos estão rigidamente formatados no site de origem.

4.2.3 Relatos extraídos do YouTube

A seguir, passamos a investigar relatos em vídeos disponíveis no YouTube, onde usuários frequentemente compartilham guias práticos demonstrando passo-a-passo como explorar falhas em dispositivos Android. Exemplos desses canais são:

- Raposo InfoCell JH[9]
- WilliansCelulares[10]

[9] <https://www.youtube.com/watch?v=CNXmt5c502g&ab_channel=RaposoInfocellJH>
[10] urlhttps://www.youtube.com/watch?v=gGQ1zxNU_nI&ab_channel=WilliansCelulares

Como esses vídeos não são exclusivamente sobre FRP Bypass, foi criado um código para filtrar os relatos de interesse com base na detecção de termos-chave que indicam possíveis vulnerabilidades relacionadas ao FRP - por exemplo, "bypass FRP", "unlock tool", "remove frp", entre outras. Esses termos-chave podem estar presentes em diferentes idiomas, como português, inglês e outros, ampliando a abrangência da detecção de vulnerabilidades a partir de diversas fontes. Ressaltamos aqui que esses termos podem ser facilmente ajustados para focar em outras vulnerabilidades de interesse, uma vez que o processamento dos textos obtidos é geral e independente da vulnerabilidade sendo monitorada.

A coleta de dados do YouTube envolve a extração de metadados, como título e descrição dos vídeos, além das transcrições automáticas. A Figura 11 traz um exemplo de vídeo que descreve como realizar o FRP Bypass em dispositivos Android.

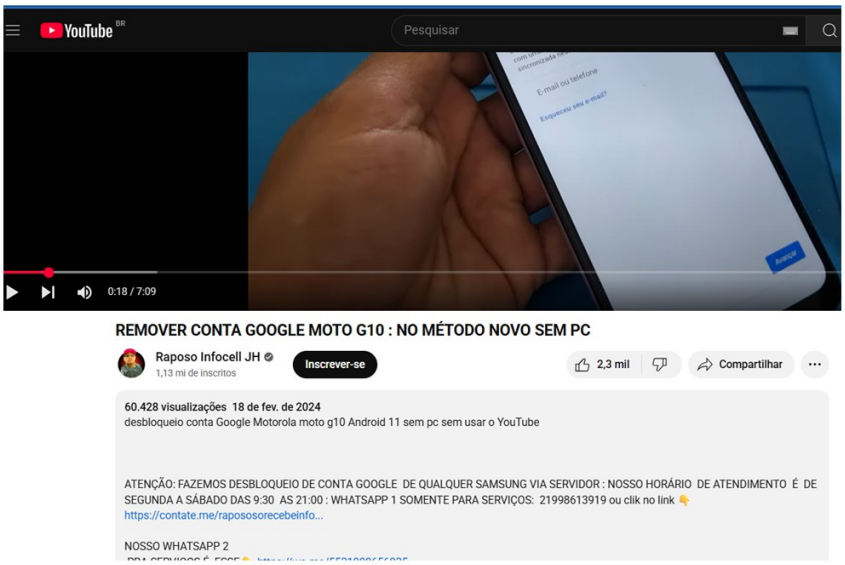

Figura 11 – Exemplo de vídeo do YouTube utilizado para extração de vulnerabilidades do Android. Fonte: Canal Raposo InfoCell JH.

A seguir, também são coletadas as transcrições automáticas dos vídeos, que são posteriormente processadas e estruturadas utilizando técnicas de PLN e LLM (ver Seção 4.3.2). Esse processo permite a análise textual das informações faladas nos vídeos, organizando o conteúdo de forma estruturada e categorizada para facilitar sua posterior interpretação e utilização.

A grande vantagem de examinar esses relatos é que eles trazem um detalhamento maior do passo-a-passo que deve ser realizado para conseguir realizar o FRP Bypass.

4.3 PRÉ-PROCESSAMENTO

As informações coletadas passam por uma etapa de pré-processamento, que é fundamental para organizar, estruturar e limpar os dados coletados, garantindo que a análise subsequente seja precisa e eficiente. Essa etapa busca normalizar os dados extraídos dos repositórios de arquivos, tutoriais e vídeos do YouTube, antes de sua análise, visto que os textos podem conter inconsistências, ruídos linguísticos e estruturas textuais desorganizadas.

Os textos coletados podem apresentar problemas como abreviações, erros ortográficos, repetições e informações irrelevantes, especialmente no caso das transcrições de vídeos do YouTube. Essas transcrições são geradas automaticamente e podem conter falhas de reconhecimento de áudio. Além disso, os vídeos frequentemente incluem informações irrelevantes, como saudações aos ouvintes, que devem ser filtradas para não interferir nas etapas futuras.

Para mitigar esses problemas, o sistema realiza a limpeza e a padronização dos textos aplicando técnicas variadas de Processamento de Linguagem Natural e de Recuperação de Informação (ver Seção 2.2). Esse processo envolve várias etapas distintas (Seção 2.2.2), que devem ser executadas de acordo com a necessidade da tarefa de mineração em foco.

No nosso caso específico, o pré-processamento vai variar de acordo com a origem dos textos sendo minerados. Os textos oriundos do site FRP Bypass, bem como a parte textual encontradas nos vídeos no YouTube passarão pelas etapas descritas na Seção 4.3.1, a seguir. Já as transcrições obtidas a partir dos vídeos no YouTube passarão pelas etapas descritas na Seção 4.3.2. Essa diferença de tratamento se deve às necessidades distintas da etapa de mineração subsequente.

4.3.1 Pré-processamento para textos originais

Os textos chamados aqui de "originais" consistem em "Título" e "Descrição" das vulnerabilidades, sendo encontrados tanto no site FRP Bypass como nas descrições dos vídeos no YouTube. Esta etapa recebe os textos sem tratamento e devolve uma lista de tokens que serão utilizados como entrada na etapa de Mineração de Dados.

- Remoção de caracteres especiais e símbolos irrelevantes – Palavras sem significado semântico, números aleatórios, URLs e caracteres especiais são eliminados para reduzir ruídos nos dados.

- Tokenização – O texto é segmentado em palavras ou frases menores para facilitar a análise e a extração de informações úteis.
- Correção de erros ortográficos – Principalmente nas transcrições automáticas, onde palavras podem ser reconhecidas de forma errada pelo sistema de reconhecimento de voz.
- Remoção de stopwords – Palavras comuns e sem valor semântico relevante, como artigos, preposições e algumas conjunções, são eliminadas para reduzir ruídos desnecessários.
- Lematização e stemming – As palavras são reduzidas às suas formas base ou raiz para uniformizar a análise. Isso ajuda a padronizar variações morfológicas, como "desbloqueando" e "desbloquear".

4.3.2 Pré-processamento para transcrições de vídeo

No caso das transcrições de vídeo, há um processo específico distinto das etapas de limpeza descritas acima, voltado para a organização inicial dos textos, que estão em formato bruto e sem estrutura (ver Figura 12). Para tornar esses textos mais úteis e interpretáveis, o sistema faz uso de Modelos de Linguagem de Grande Escala (LLMs), como OpenAI ChatGPT e Google Gemini, que auxiliam na reestruturação e segmentação do conteúdo obtido (ver Figura 13). Os modelos generativos são utilizados para:

- Organizar as transcrições de vídeos em descrições mais estruturadas e legíveis, eliminando gírias e frases desnecessárias ou repetitivas.
- Converter as transcrições em guias passo-a-passo, quando aplicável, facilitando a compreensão dos métodos explorados nos vídeos.
- Resumir informações redundantes e destacar os pontos mais relevantes, tornando a análise mais eficiente.

Esse processo permite que os textos extraídos dos vídeos do YouTube sejam transformados em informações organizadas e interpretáveis, facilitando sua posterior análise no sistema.

Ao final do pré-processamento, os dados coletados e limpos são passados para o próximo módulo, que envolve mineração de dados e análise de similaridade. Esse processo é crucial para garantir que apenas informações relevantes sejam utilizadas na detecção de vulnerabilidades,

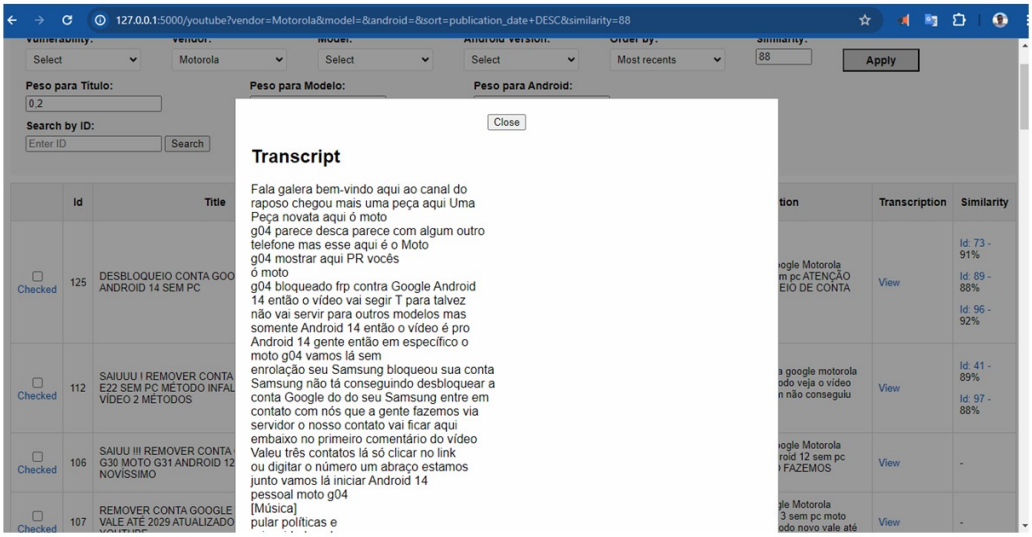

Figura 12 – Exemplo de transcrição bruta.

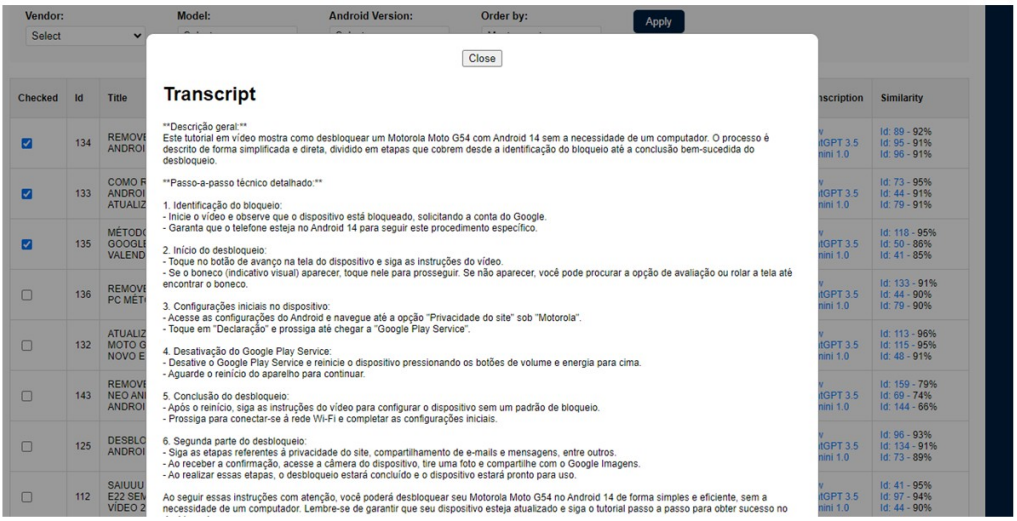

Figura 13 – Um exemplo de transcrição corrigida.

evitando a análise de dados irrelevantes ou incorretos. A Seção 5.1.4.2 traz detalhes sobre a implementação desse módulo de pré-processamento.

4.4 MINERAÇÃO DE DADOS

Após o pré-processamento, os dados estruturados são submetidos à etapa de Mineração de Dados e Análise de Similaridade. O principal objetivo dessa fase é extrair padrões, correlacionar relatos coletados e identificar novas vulnerabilidades que possam estar relacionadas a casos já conhecidos. Dessa forma, o sistema permite analisar e comparar as informações extraídas, proporcionando percepções valiosas sobre a recorrência e relevância das vulnerabilidades identificadas.

Essa etapa consiste na extração de conhecimento útil a partir dos relatos coletados. Para isso, são aplicadas técnicas que permitem identificar padrões, categorizar informações e correlacionar novos relatos com vulnerabilidades previamente documentadas. Esse processo envolve as seguintes etapas:

- Etiquetagem dos relatos – Após a coleta e limpeza, os textos passam por um processo de etiquetagem automática, no qual são extraídos metadados fundamentais para sua contextualização. Essas etiquetas incluem metadados, como fabricante, modelo e versão do Android, além de termos associados à vulnerabilidade descrita. As etiquetas são obtidas a partir do próprio conteúdo textual e, quando necessário, complementadas com informações provenientes do site de origem ou da fonte de dados. As tags são essenciais para organizar e filtrar os relatos. Elas permitem associar cada relato à vulnerabilidade descrita, ao modelo ou à versão do Android, facilitando a busca e agrupamento de relatos semelhantes, e, por consequência, a análise de similaridade. Uma vez extraídas, essas etiquetas são associadas ao relato no momento de seu armazenamento no banco de dados relacional MySQL, conforme descrito na Seção 5.1.4.4. Esse modelo relacional facilita consultas posteriores e permite uma análise segmentada e eficaz por parte dos especialistas de segurança.

- Cálculo da similaridade - Após a etiquetagem, os relatos seguem para a etapa de análise de similaridade, que busca avaliar a correlação entre novos relatos e vulnerabilidades previamente identificadas. Essa análise é realizada utilizando métricas de similaridade como a Distância de Levenshtein e modelos de Similaridade Semântica (SBERT), conforme detalhado na Seção 4.4.1.

Antes da comparação, os metadados como fabricante, modelo e versão do Android são

	source_id	model_names	vendor_name	android_versions	transcript
elp you to Regain acce...	bypassfrpfiles	Moto G35	Motorola	14	NULL
verification screen on ...	bypassfrpfiles	Moto G45	Motorola	14	NULL
	youtube	Moto E22	Motorola	12	fala galerinha bem-
	youtube	Moto G10	Motorola	11	fala galerinha bem-
	youtube	Moto E6i	Motorola	10	e fala galerinha ber
	youtube	Moto G50	Motorola	12	e fala galerinha tud
	youtube	Moto G24,Moto G34,M...	Motorola	13	Fala galera bem-vir
	youtube	Moto G7 Play	Motorola	10	Fala galera bem-vir
	youtube	Moto G04,Moto G04s	Motorola	14	fala galerinha bem-
	youtube	Edge 30 Neo,Edge 40,...	Motorola	-	Fala galera bem-vir
	youtube	Moto E13	Motorola	-	fala galerinha quan
	youtube	Moto G41	Motorola	12	fala galerinha bem-
	youtube	Moto G32	Motorola	12,13	vamos lá vamos lá e
	youtube	Moto E32	Motorola	11	fala galerinha hoje
	youtube	One Vision	Motorola	11	Fala galera moto or
	youtube	Moto E7 Plus	Motorola	-	fala galerinha vamo
	youtube	Moto E20	Motorola	11	fala galerinha bem-

Figura 14 – Um exemplo de tabela do BD com as tags.

removidos das frases. Isso garante que o cálculo de similaridade leve em consideração apenas o conteúdo textual relevante. A métrica de Levenshtein é aplicada diretamente sobre frases completas, como títulos e descrições curtas, sem segmentação em palavras individuais. Esse procedimento resulta em uma medida global de diferença entre os textos, permitindo detectar redundâncias e priorizar relatos com conteúdo inédito.

Essa etapa é fundamental para agrupar relatos semelhantes, evitar duplicações de esforço pelas equipes de segurança e destacar novos casos de interesse.

4.4.1 Análise de Similaridade

Um dos problemas críticos enfrentados pela empresa parceira no processo manual de busca por relatos de vulnerabilidades é a redundância de informação na Web. Sempre que um texto é coletado, um colaborador da equipe de segurança da Motorola necessita analisar seu conteúdo e reportar os problemas identificados ao time de desenvolvimento.

Para mitigar esse problema, o sistema desenvolvido realiza automaticamente a mensuração da similaridade entre os relatos. A cada novo registro inserido, o sistema o compara com todos os relatos previamente armazenados no banco de dados, indicando se o conteúdo representa uma vulnerabilidade já conhecida ou se traz informações inéditas (ver Figura 15).

Assim como ocorre na etapa de pré-processamento (Seção 4.3), o cálculo de similaridade também é diferente para cada tipo de texto coletado. Os textos oriundos do site FRP Bypass e as descrições textuais obtidas dos vídeos do YouTube serão comparados através da métrica conhecida como "Distância de Levenshtein" (Seção 4.4.1.1). Já os relatos oriundos do YouTube

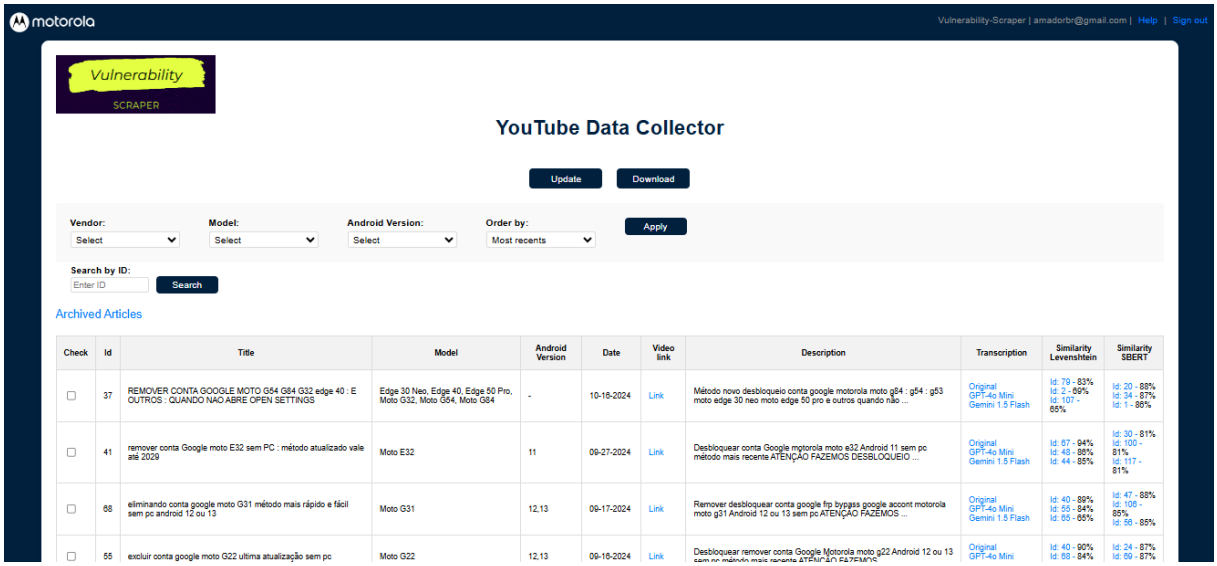

Figura 15 – Tela de visualização para análise dos resultados de similaridade (imagem extraída do protótipo). Fonte: Autor.

são submetidos a uma análise semântica via SBERT (Seção 4.4.1.2).

4.4.1.1 Comparação textual com Distância de Levenshtein

Esta etapa mede a semelhança entre novos relatos e os dados já armazenados no sistema utilizando métricas de *distância textual*. Utilizamos aqui a *Distância de Levenshtein* (ver Seção 2.2.4.1.1).

Essa métrica calcula o número mínimo de operações necessárias para transformar uma dada string em outra, considerando inserções, deleções e substituições de tokens. Esse valor representa o grau de semelhança entre duas expressões, sendo menor quanto mais parecidas forem as frases comparadas.

No sistema proposto, esse cálculo é aplicado separadamente a três elementos-chave extraídos de cada relato:

- Título da vulnerabilidade;
- Modelo do dispositivo Android;
- Versão do sistema operacional Android.

Cada uma dessas comparações gera um valor de similaridade individual. Em seguida, os três valores são combinados por meio de uma média ponderada, utilizando pesos definidos

empiricamente de acordo com a importância relativa de cada campo na identificação de vulnerabilidades. Os pesos adotados (10% para título, 40% para modelo e 50% para versão do Android) foram ajustados a partir de testes com diferentes configurações, até que o sistema apresentasse um ranking considerado satisfatório pelo Product Owner (P.O.) da empresa parceira, equilibrando precisão e relevância dos resultados obtidos.

Essa ponderação busca garantir que os campos mais determinantes, especialmente o modelo do aparelho e a versão do Android, influenciem mais fortemente no resultado final de similaridade. O campo do "título", embora relevante, recebe menor peso devido à sua maior variabilidade linguística.

Por exemplo, considere os seguintes de relatos:

1. "Nova técnica de desbloqueio FRP para o Moto G100 sem necessidade de PC."

2. "Método atualizado para remover o FRP do Moto G100 sem usar computador."

Apesar de não serem idênticos, a presença de termos comuns como "FRP", "Moto G100" e "desbloqueio" indica forte similaridade entre os textos. Mesmo com variações na forma de escrita, a métrica de Levenshtein resultou em um valor de 0,3662, o que representa alguma similaridade textual e confirma a proximidade entre os relatos, ainda que com diferenças na estrutura das frases.

Vale destacar que, embora o sistema também armazene termos-chave relacionados ao FRP (como "bypass FRP", "unlock tool", "remove frp"), esses termos não são diretamente utilizados no cálculo da métrica de Levenshtein. A métrica é aplicada exclusivamente sobre os campos estruturados do relato — título, modelo e versão —, garantindo uma base consistente para o cálculo.

Como já visto, o sistema aplica etapas de pré-processamento aos textos antes do cálculo, removendo artigos, preposições, conjunções e variações morfológicas, o que contribui para a redução de ruídos e melhora a precisão do resultado. Essa estratégia permite detectar relatos semelhantes, mesmo com variações linguísticas, abreviações ou pequenos erros de digitação, contribuindo para uma análise mais precisa e eficiente das vulnerabilidades descritas.

Essa abordagem permite que relatos com variações ortográficas, abreviações ou expressões equivalentes sejam corretamente agrupados, favorecendo a detecção de duplicatas e a priorização de relatos inéditos. Como resultado, a análise se torna mais eficiente, precisa e menos suscetível a redundâncias.

A Seção 5.1.4.3 do Capítulo 5 detalha a implementação prática da análise de similaridade com Levenshtein.

4.4.1.2 Análise semântica com SBERT

Com a evolução do trabalho, tornou-se evidente a necessidade de empregar técnicas mais avançadas de comparação textual, capazes de ir além da simples igualdade entre palavras. Muitas vezes, relatos sobre vulnerabilidades são descritos com palavras diferentes, mas carregam o mesmo significado essencial. Para capturar essas nuances, foi adotado o modelo SBERT (Sentence-BERT), baseado em aprendizado profundo e projetado para realizar análises semânticas de sentenças (ver Seção 2.2.4.1).

Ao contrário de métodos tradicionais, como a distância de Levenshtein, que comparam textos com base em operações sobre caracteres, o SBERT transforma sentenças em vetores numéricos de alta dimensão (*embeddings*). Essa representação permite medir a similaridade semântica entre frases, mesmo que elas utilizem vocabulário distinto. Assim, palavras sinônimas ou contextualmente relacionadas são reconhecidas como semanticamente próximas dentro do espaço vetorial.

Considere os seguintes relatos sobre desbloqueio de FRP em dispositivos Motorola:

1. "Nova técnica de desbloqueio FRP para o Moto G100 sem necessidade de PC."
2. "Método atualizado para remover o FRP do Moto G100 sem usar computador."

Embora diferentes na redação, ambas descrevem o mesmo conceito. Nesse exemplo, o valor da similaridade entre as frases foi de 71.04%. O SBERT captura essa similaridade semântica, garantindo que relatos equivalentes sejam identificados e agrupados corretamente.

Assim, vemos que o SBERT apresenta muitas vantagens quando comparado à distância de edição: é capaz de detectar relatos com o mesmo significado, ainda que redigidos de forma diferente; reduz falsos negativos, agrupando informações relevantes; e melhora a organização e a priorização das vulnerabilidades, facilitando a análise pela equipe de segurança da Motorola.

Com essa abordagem, o sistema tornou-se mais robusto e inteligente na detecção, organização e priorização de relatos de vulnerabilidade. O resultado é uma análise mais confiável e menos suscetível às limitações linguísticas dos relatos encontrados na Web.

A Seção 5.1.4.3 do Capítulo 5 apresenta mais detalhes sobre a implementação dessa etapa baseada em SBERT.

4.4.1.3 Ranking de relevância

Por fim, os resultados do processo de mineração são ranqueados, para serem exibidos através da interface do sistema. Por padrão, os novos relatos sempre ficam no topo da lista (tela de resultados ver Figura 15). Isso ajuda na priorização das informações mais relevantes, permitindo que especialistas se concentrem primeiro nos relatos com maior potencial de representar uma nova vulnerabilidade crítica.

Ao final da etapa de Mineração de Dados, os novos relatos já estão organizados e associados a vulnerabilidades conhecidas através do percentual de similaridade, permitindo que o sistema forneça uma base sólida para análise e tomada de decisão. A próxima etapa do fluxo é o armazenamento dos dados, garantindo que todas as informações processadas fiquem disponíveis para consultas futuras e visualização na interface do sistema.

4.5 ARMAZENAMENTO DOS DADOS

Após a coleta, o pré-processamento e a mineração de dados, as informações extraídas passam pela etapa de armazenamento e organização, onde são estruturadas em um banco de dados relacional para facilitar a consulta e posterior análise. Esse processo garante que os dados coletados fiquem acessíveis de forma eficiente, permitindo a recuperação e a visualização de relatos de vulnerabilidades de maneira estruturada.

O banco de dados armazena diversas informações essenciais para a detecção de vulnerabilidades, incluindo textos extraídos de repositórios e vídeos do YouTube, termos identificados no pré-processamento, similaridades calculadas entre novos relatos e vulnerabilidades já conhecidas, além das associações com fabricantes, modelos e versões do Android.

Para garantir a organização dos dados, cada relato armazenado é vinculado automaticamente a metadados relevantes, como:

- Origem da informação (e.g., repositório de arquivos, fórum técnico, vídeo do YouTube);
- Palavras-chave identificadas (e.g., fabricante, modelo, versão do Android, termos específicos de FRP);

- Texto processado e normalizado, facilitando futuras análises;
- Similaridade com relatos anteriores, permitindo uma busca eficiente por vulnerabilidades semelhantes.

Essa estruturação permite que os dados sejam facilmente acessados e analisados posteriormente através da interface Web, proporcionando aos especialistas um meio ágil de visualizar tendências, padrões e recorrência de vulnerabilidades. A estrutura do banco de dados permite uma possível expansão da solução para outras vulnerabilidades além do FRP, garantindo flexibilidade para futuras melhorias.

4.6 VISUALIZAÇÃO E ANÁLISE DOS RESULTADOS

O módulo final do processo consiste na interface Web do sistema, através da qual os dados coletados e tratados poderão ser gerenciados e analisados pelos especialistas de forma eficiente. Os colaboradores podem utilizar essas informações para tomada de decisão e investigação detalhada das vulnerabilidades detectadas. A interface foi projetada para oferecer uma experiência intuitiva, com filtros, ferramentas de pesquisa e funcionalidades interativas, otimizando a análise e priorização de vulnerabilidades.

Como pode ser visto na Figura 15, através da Interface os usuários podem visualizar os relatos de vulnerabilidades coletados com informações como sua origem, os modelos de dispositivos e versões do Android afetados, além do valor de similaridade de cada relato em relação aos outros dados já existentes, permitindo comparar rapidamente novas vulnerabilidades com aquelas já conhecidas. Como já mencionado, os novos relatos sempre ficam no topo da lista.

4.6.1 Funcionalidades da Interface Web

A Figura 16 é uma cópia da Figura 15, e foi repetida aqui para facilitar a explicação das funcionalidades da interface do sistema.

- Botão de atualização ("Update"): Quando acionado, executa uma nova varredura nos repositórios de dados (ex: YouTube), garantindo que novas informações sejam coletadas e processadas.

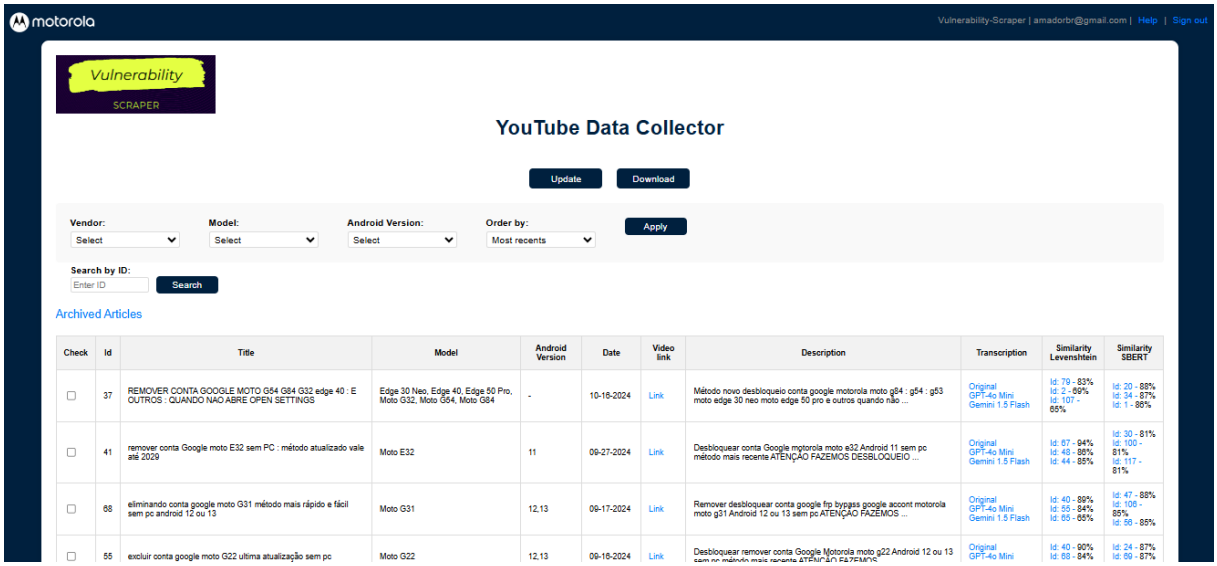

Figura 16 – Tela de visualização para análise dos resultados de similaridade (imagem extraída do protótipo). Fonte: Autor.

- Filtros avançados: Permitem filtrar os relatos por fabricante, modelo, versão do Android e também ordená-los pelos mais recentes, proporcionando uma análise direcionada.
- Marcação como visualizado: Cada postagem pode ser marcada como visualizada por meio de um checkbox. A partir dessa marcação, o item será automaticamente ocultado da tela principal, porém esse item não será excluído do BD. Os itens marcados como visualizados permanecem acessíveis em uma página separada, intitulada "Artigos Visualizados".
- Pesquisa direta por ID: Possibilidade de localizar um relato específico por meio do seu identificador único.
- Adição de comentários: Os especialistas podem adicionar comentários em cada postagem, permitindo um melhor acompanhamento e troca de informações entre colaboradores da equipe sobre cada vulnerabilidade detectada.
- Exportação de dados: A interface possui um botão para download dos relatos em formato CSV, permitindo a exportação dos dados para análises externas ou documentação.
- Segurança e controle de acesso: Para garantir a segurança dos dados, a autenticação da interface é realizada via OAuth (Google Login), restringindo o acesso apenas a usuários autorizados. Esse controle evita acessos não autorizados a informações sensíveis e garante que apenas membros da equipe de segurança possam gerenciar os relatos identificados.

Essa interface web centraliza e organiza as vulnerabilidades detectadas, permitindo que especialistas conduzam investigações detalhadas e atuem de forma proativa na mitigação de riscos. Ao oferecer funcionalidades como filtros avançados, pesquisa por ID, atualização automática e exportação de dados, a solução garante maior eficiência e precisão na análise das vulnerabilidades do Android.

Como trabalhos futuros, será desenvolvido um controle de Login personalizado, de modo que será possível armazenar no BD as ações de cada usuário individualmente. Assim, será possível rastrear quem já acessou cada relato e clicou no checkbox. Além disso, através desse controle será possível atribuir permissões de acesso distintas para cada usuário, afetando as possibilidades de editar o arquivo de configurações, por exemplo.

4.7 CONSIDERAÇÕES FINAIS

Este capítulo apresentou detalhadamente a solução proposta para identificação e extração de vulnerabilidades em dispositivos Android, com foco inicial no problema de FRP Bypass. A abordagem desenvolvida combina técnicas de Web Scraping, Processamento de Linguagem Natural, Modelos de Linguagem e Mineração de Dados, estruturando um fluxo que parte da coleta de relatos online até a exibição dos resultados por meio de uma interface interativa.

Foram descritos todos os módulos implementados, incluindo a coleta de dados em múltiplas fontes (sites especializados e YouTube), o pré-processamento textual, a análise de similaridade (textual e semântica), a organização e armazenamento em banco de dados e, por fim, a visualização dos resultados. Destaca-se a aplicação conjunta de métricas tradicionais (como Levenshtein) e atuais (como SBERT), que conferem maior precisão à identificação de relatos redundantes ou inéditos.

A solução permite maior agilidade e eficiência na detecção de novas vulnerabilidades, reduzindo o esforço manual exigido anteriormente pela equipe da empresa parceira. Além disso, o sistema é flexível e pode ser adaptado para novos tipos de vulnerabilidades ou fabricantes.

Capítulo 5 detalha a implementação do protótipo e os testes realizados para validar sua eficácia.

Além disso, destaca-se que o processo de coleta foi concebido de forma configurável, permitindo ao usuário definir os canais e fontes de interesse no YouTube, garantindo maior controle sobre a relevância e a confiabilidade dos dados coletados. Por fim, a escolha pelo uso combinado das métricas de similaridade — Levenshtein para textos curtos e estruturados e

SBERT para textos extensos e semânticos — buscou equilibrar precisão sintática e abrangência semântica, tornando o sistema mais robusto e eficiente na identificação de vulnerabilidades.

5 O SISTEMA PROTÓTIPO IMPLEMENTADO: TESTES E RESULTADOS

A complexidade crescente das ameaças cibernéticas no ecossistema Android exige abordagens automatizadas e escaláveis para identificação e mitigação de vulnerabilidades. Neste contexto, com base nos requisitos estabelecidos no Capítulo 4, este trabalho apresenta a implementação, validação e avaliação experimental de um sistema protótipo para identificação, extração e análise de vulnerabilidades em dispositivos Android, com ênfase inicial na funcionalidade *Factory Reset Protection* (FRP).

O protótipo foi concebido com arquitetura modular e semi-automatizada, integrando técnicas de Web Scraping, Processamento de Linguagem Natural, Modelos de Linguagem (LLMs) e algoritmos de similaridade textual. O fluxo do sistema abrange desde a coleta de dados em múltiplas fontes públicas, passando pelo pré-processamento textual, análise semântica e mineração de similaridade, até a disponibilização dos resultados por meio de uma interface web interativa e segura.

As etapas de coleta e limpeza de dados, por opção de controle e rastreabilidade, são acionadas manualmente via interface web, conferindo flexibilidade e supervisão ao processo. Todas as decisões metodológicas, bem como a estruturação dos módulos, foram pautadas na viabilidade de expansão futura do sistema para outros tipos de vulnerabilidades além do FRP.

Este capítulo detalha a implementação dos módulos do protótipo (Seção 5.1), os testes realizados (Seção 5.2) e os principais resultados experimentais (Seção 5.3), sempre alinhando os aspectos técnicos à motivação original do projeto.

Por fim, vale destacar que, para fins deste trabalho, todos os relatos extraídos de sites, fóruns ou vídeos são denominados "artigos", conforme convenção adotada na área e padronizada neste trabalho.

5.1 IMPLEMENTAÇÃO DO PROTÓTIPO

A implementação do sistema adotou uma arquitetura modular, orientada à escalabilidade, manutenibilidade e evolução contínua. O protótipo foi concebido para orquestrar, de forma semi-automatizada, todas as etapas do pipeline de detecção de novas vulnerabilidades, permitindo a substituição ou aprimoramento de componentes individuais sem impactar o restante da estrutura.

5.1.1 Arquitetura do Software

O sistema foi projetado com arquitetura modular, voltada à escalabilidade, facilidade de manutenção e evolução incremental. Essa estrutura orquestra de forma semi-automatizada todas as etapas do pipeline de identificação de vulnerabilidades — desde a coleta de dados até a análise e visualização —, permitindo a atualização de módulos individuais sem comprometer a estabilidade geral da solução.

O sistema é composto por cinco módulos principais, com responsabilidades claramente delimitadas:

1. **Coleta de Dados**: Realiza a extração automatizada de informações a partir de fontes abertas – como o site BypassFRPFiles, o GSMArena e canais do YouTube. Esta etapa é executada por scripts Python especializados (smartphones_models_scraping.py, webscraping_monograph.py, google_hacking.py). Essa etapa contempla técnicas de web scraping, integração com APIs e buscas automatizadas. O início do processo pode ser manual, via interface web, conforme as necessidades de controle e rastreamento.

2. **Pré-processamento com PLN**: Após a coleta, os textos passam por uma etapa de limpeza, normalização e estruturação. Técnicas de Processamento de Linguagem Natural (PLN) são aplicadas para eliminar ruídos, tokenizar, lematizar e padronizar o conteúdo. Transcrições de vídeo são reestruturadas por modelos de linguagem (ChatGPT[1] ou Gemini[2]) para garantir clareza técnica.

3. **Mineração de Dados e Análise de Similaridade**: Esta camada utiliza algoritmos de extração de termos, categorização semântica e comparação textual (Distância de Levenshtein para textos curtos e SBERT para comparações semânticas) para identificar duplicidades, padrões e conteúdos correlatos.

4. **Armazenamento e Organização**: Todas as informações são persistidas em um banco de dados relacional MySQL[3], intermediadas por uma camada de acesso (Data Access Layer – DAL) composta por classes como ArticleDAL, ModelDAL, VendorDAL, SimilarityDAL e ArticleViewDAL. Isso assegura integridade referencial, consultas eficientes e facilita futuras expansões.

[1] https://chatgpt.com/
[2] https://gemini.google.com/?hl=pt-BR
[3] https://www.mysql.com/

5. **Interface Web para Análise e Visualização**: Desenvolvida em Flask[4], a interface web permite aos analistas da Motorola visualizar, filtrar, marcar, comentar e exportar artigos. Toda interação com o banco de dados é mediada por views e rotas seguras, com autenticação OAuth restrita ao domínio @motorola.com.

A Figura 17 apresenta a arquitetura em camadas do sistema, destacando a organização modular e o fluxo de dados entre os componentes. Cada módulo opera de forma independente, viabilizando evolução incremental, integração de novas fontes e ajustes pontuais para diferentes tipos de vulnerabilidades além do FRP.

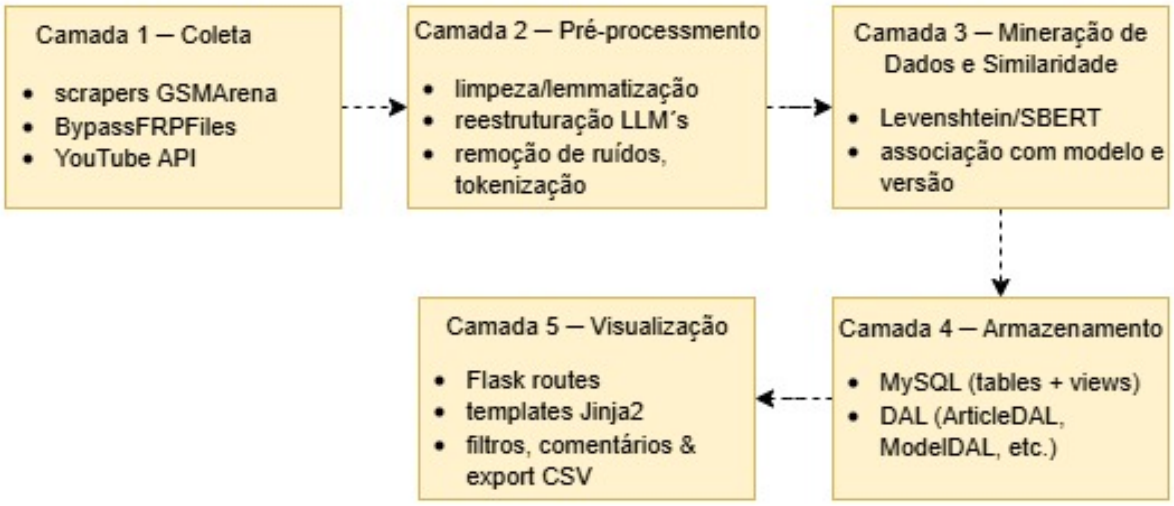

Figura 17 – Diagrama de Camadas do sistema, evidenciando a organização modular e o fluxo de dados entre os componentes. Fonte: Autor.

5.1.2 Linguagem, frameworks e bibliotecas utilizadas

A implementação do protótipo foi realizada predominantemente em Python[5], dada sua expressiva comunidade, abundância de bibliotecas especializadas para análise de dados, mineração de texto, integração com APIs e automação de tarefas. O Python também oferece excelente suporte para construção de aplicações web leves, o que favoreceu a integração eficiente entre os diversos módulos do sistema.

No total, o sistema desenvolvido contém aproximadamente 5.233 linhas de código, distribuídas entre os módulos de coleta, processamento, análise de similaridade, banco de dados

[4] https://flask.palletsprojects.com/
[5] https://www.python.org/

e interface web. A Tabela 1 apresenta as principais tecnologias adotadas e suas funções no pipeline do sistema:

Tabela 1 – Principais tecnologias utilizadas no sistema

Categoria	**Tecnologias**
Desenvolvimento Web e Interface	Flask, Jinja2[6], HTML[7], CSS[8], JavaScript[9]
Coleta de Dados	requests[10], BeautifulSoup 4[11], YouTube Data API v3[12]
Processamento de Texto e PLN	html2text[13], re (Regex)[14], spaCy[15], OpenAI API[16], Vertex AI[17]
Mineração de Dados	python-Levenshtein[18], sentence-transformers[19]
Banco de Dados	MySQL, mysql-connector-python[20]
Exportação e Relatórios	panda[21]

- **Desenvolvimento Web e Interface**
  - **Flask**: Microframework utilizado para estruturar o backend da aplicação, definir rotas, autenticação OAuth e integração entre módulos.
  - **Jinja2**: Engine de templates empregada na renderização dinâmica dos arquivos HTML (armazenados em /templates).
  - **HTML, CSS e JavaScript**: Fundamentais para a construção da interface web responsiva, garantindo interatividade e usabilidade.
- **Coleta de Dados (Web Scraping e APIs)**

[6] https://jinja.palletsprojects.com/
[7] https://html.spec.whatwg.org/
[8] https://www.w3.org/Style/CSS/
[9] https://developer.mozilla.org/docs/Web/JavaScript
[10] https://requests.readthedocs.io/
[11] https://www.crummy.com/software/BeautifulSoup/
[12] https://developers.google.com/youtube/v3
[13] https://pypi.org/project/html2text/
[14] https://docs.python.org/3/library/re.html
[15] https://spacy.io/
[16] https://platform.openai.com/docs/
[17] https://cloud.google.com/vertex-ai
[18] https://pypi.org/project/python-Levenshtein/
[19] https://www.sbert.net/
[20] https://dev.mysql.com/doc/connector-python/en/
[21] https://pandas.pydata.org/

  - **requests**: Realiza requisições HTTP para extração de páginas web e comunicação com APIs públicas.
  - **BeautifulSoup 4 (bs4)**: Utilizada para o parsing e extração de dados estruturados do HTML.
  - **YouTube Data API v3**: Responsável por buscas automatizadas e coleta de metadados de vídeos relevantes.
  - **youtube_transcript_api**[22]: Automatiza a obtenção de transcrições textuais de vídeos do YouTube.

- **Processamento de Texto e PLN**
  - **html2text**: Converte conteúdo HTML em texto puro, facilitando o pré-processamento.
  - **re (Regex)**: Realiza limpeza textual, remoção de padrões indesejados e ruídos.
  - **spaCy**: Empregado para lematização, tokenização e análise linguística.
  - **OpenAI API**: Utilizada para reestruturação e padronização técnica de transcrições de vídeos, por meio de prompts customizados.
  - **Vertex AI (Google Gemini)**: Alternativa de LLM para reestruturação de textos via API.

- **Mineração de Dados e Análise de Similaridade**
  - **python-Levenshtein**: Empregado na comparação de textos curtos, como títulos e descrições.
  - **sentence-transformers**: Biblioteca baseada em SBERT, para geração de embeddings semânticos e cálculo de similaridade vetorial.

- **Banco de Dados e Persistência**
  - **MySQL**: Sistema gerenciador relacional utilizado para armazenar fabricantes, modelos, artigos, transcrições e índices de similaridade.
  - **mysql-connector-python**: Integração entre scripts Python e o banco MySQL.

[22] https://pypi.org/project/youtube-transcript-api/

  - **python-decouple**[23] e **dotenv**[24]: Gerenciam variáveis de ambiente sensíveis, como credenciais, tokens e parâmetros de configuração.

- **Exportação e Relatórios**

  - **pandas**: Utilizada para manipulação de dados tabulares e exportação de relatórios em formato CSV via interface web.

Essa combinação de ferramentas, bibliotecas e frameworks propiciou uma base tecnológica robusta, modular e extensível, capaz de suportar tanto as demandas presentes quanto futuras expansões do sistema para novas fontes de dados ou técnicas de análise.

5.1.3 Organização do código e estrutura dos módulos

A estruturação do código do sistema foi concebida para garantir modularidade, desacoplamento e fácil manutenção, refletindo o princípio de separação de responsabilidades. Cada etapa central do *pipeline* — desde a coleta, passando pelo pré-processamento, análise e visualização — está encapsulada em módulos independentes, facilitando sua evolução, testes e reuso.

A seguir, é descrita a organização dos principais componentes do projeto (ver Figura 18).

- **Raiz do Projeto (/)** — Concentra os scripts principais do pipeline, entre eles:

  - app.py: Responsável por orquestrar a aplicação Flask, integrando rotas da interface web, autenticação OAuth e a execução sequencial dos módulos de coleta, pré-processamento, mineração e armazenamento dos artigos.
  - database_creation.py: Automatiza a criação das tabelas, views e chaves do banco de dados relacional MySQL.
  - smartphones_models_scraping.py: Realiza scraping dos modelos de dispositivos no GSMArena.
  - webscraping_monograph.py: Executa a coleta de tutoriais técnicos no BypassFRP-Files.com.
  - google_hacking.py: Realiza busca e coleta de metadados de vídeos, utilizando técnicas de Google Hacking e a API do YouTube.

[23] https://pypi.org/project/python-decouple/
[24] https://pypi.org/project/python-dotenv/

  - Extractor.py: Realiza o pré-processamento textual, aplicando técnicas de limpeza, normalização e lematização nos artigos coletados.
  - similarity.py: Responsável pelo cálculo de similaridade textual entre artigos, por meio dos algoritmos Levenshtein e SBERT.
  - transcript.py e transcript_generative.py: Gerenciam a extração de transcrições de vídeos e a reestruturação desses textos por modelos de linguagem natural.
  - Outros scripts de configuração e integração, como arquivos .env, chaves de API e arquivos de inicialização.

- **/database/** — Contém todas as classes responsáveis pelo acesso e manipulação das tabelas do banco de dados (Data Access Layer — DAL). Exemplos: ArticleDAL.py, ModelDAL.py, VendorDAL.py, SimilarityDAL.py, ArticleViewDAL.py.
- **/templates/** — Armazena os arquivos HTML dos templates de interface, construídos com Jinja2. Destacam-se os arquivos youtube_listed.html, bypassfrpfiles_listed.html, show_article.html e index.html.
- **/utils/** — Inclui scripts auxiliares, como o sources.py, com funções e constantes de apoio para as etapas de coleta e processamento.
- **/data/** — Contém arquivos de configuração e apoio às buscas (como o search.json, que armazena termos e parâmetros para o Google Hacking).

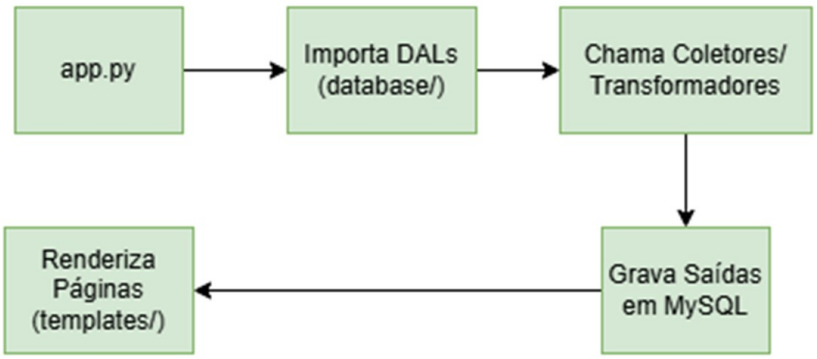

Figura 18 – Fluxo de integração entre os principais módulos do sistema. Fonte: Autor.

O fluxo do sistema é orientado pela aplicação principal (app.py), que importa as classes da camada DAL (localizadas em /database/) e aciona, conforme necessidade, os coletores, processadores e analisadores implementados como scripts independentes na raiz do projeto. Após o processamento, os dados resultantes são persistidos no banco MySQL, e, em seguida, renderizados para o usuário por meio dos templates HTML em /templates/. Os dados intermediários trafegam predominantemente como dicionários Python e/ou objetos JSON, mantendo as dependências mínimas e favorecendo uma arquitetura desacoplada.

Essa organização modular e estruturada oferece múltiplos benefícios:

- Desacoplamento: Mudanças em um módulo (por exemplo, atualização de técnicas de mineração ou de scraping) não afetam as demais etapas do sistema.
- Reutilização: Scripts, funções e classes podem ser facilmente reaproveitados ou adaptados para outros projetos, facilitando a expansão para novos domínios.
- Testabilidade: Cada componente pode ser testado isoladamente, garantindo maior robustez e rastreabilidade durante o ciclo de vida do desenvolvimento.
- Extensibilidade: Novas fontes de dados e funcionalidades podem ser integradas com baixo impacto sobre a arquitetura existente.

A execução dos módulos não segue obrigatoriamente uma sequência fixa. O processo de coleta pode ocorrer de forma independente e recorrente, alimentando o banco de dados periodicamente, enquanto as etapas de pré-processamento e análise de similaridade são acionadas conforme a chegada de novos dados. Essa abordagem garante maior flexibilidade e aproveitamento do tempo de processamento, já que o tempo de coleta tende a ser superior ao das demais etapas.

Assim, a estrutura modular e bem organizada do projeto contribui para a manutenção, evolução contínua e rápida adaptação a mudanças tecnológicas ou de requisitos emergentes, mantendo o alinhamento com as melhores práticas de engenharia de software em projetos de pesquisa aplicada. O fluxo de integração entre os principais módulos do sistema pode ser visualizado na Figura 18.

5.1.4 Implementação dos Módulos do Sistema

A implementação do sistema protótipo foi organizada em módulos independentes, refletindo as etapas fundamentais do pipeline de identificação automatizada de vulnerabilidades em dispositivos Android. Cada módulo foi desenvolvido para executar uma função específica, mantendo a coesão interna e o desacoplamento entre as diferentes camadas do sistema.

A seguir, são descritos os principais módulos que compõem o sistema, abordando as soluções técnicas adotadas, as decisões de projeto e o alinhamento com os requisitos funcionais apresentados no Capítulo 4: Coleta de Dados (Seção 5.1.4.1), Pré-processamento (Seção 5.1.4.2) , Mineração de Dados e Análise de Similaridade (Seção 5.1.4.3), Armazenamento e Organização (Seção 5.1.4.4), e Interface Web para Análise e Visualização (Seção 5.1.4.5).

Cada um desses módulos será detalhado nas seções a seguir, evidenciando tanto as decisões técnicas quanto a relação entre a arquitetura proposta e as necessidades da análise de segurança em dispositivos Android.

5.1.4.1 Coleta de Dados

A etapa de coleta de dados constitui o ponto inicial do pipeline do sistema, sendo fundamental para garantir a abrangência e a representatividade do conjunto de artigos analisados (tanto textos técnicos quanto vídeos do YouTube). Esta etapa foi estruturada para possibilitar a extração automatizada e incremental de dados relevantes de diferentes fontes públicas, respeitando critérios de atualização, rastreabilidade e flexibilidade.

A arquitetura do sistema contempla três submódulos de coleta:

- **Coleta de Fabricantes e Modelos (GSMArena):** O script smartphones_models_scraping.py realiza o web scraping no site GSMArena, extraindo a lista de fabricantes e modelos de dispositivos Android. Esses dados são utilizados como referência para a categorização dos artigos e para compor as tabelas vendor e model no banco de dados.
- **Coleta de Artigos Textuais (BypassFRPFiles.com):** O módulo webscraping_monograph.py automatiza a extração de tutoriais técnicos publicados no site BypassFRPFiles.com. O fluxo abrange a identificação de artigos relevantes, o acesso ao conteúdo completo das páginas e o armazenamento dos metadados e textos na tabela article, atribuindo a origem como "bypassfrpfiles". Os dados coletados são processados em etapas posteriores,

sem integração direta à camada de persistência no momento da coleta, respeitando o princípio de desacoplamento entre coleta e armazenamento.

- **Coleta de Metadados e Artigos em Vídeo (YouTube):** O script google_hacking.py utiliza dorks configurados em search.json e a API do YouTube para identificar vídeos relacionados a vulnerabilidades Android, especialmente métodos de desbloqueio FRP. São extraídos metadados como título, link, descrição, data de publicação e identificador do vídeo. Em seguida, o módulo transcript.py tenta obter as transcrições dos vídeos via youtube_transcript_api. Entretanto, nem todos os vídeos possuem transcrições disponíveis devido a limitações técnicas da plataforma ou configurações do canal. Quando a transcrição não está disponível, o sistema armazena apenas os metadados, mantendo rastreabilidade das tentativas de extração.

A arquitetura da coleta de dados foi projetada para garantir flexibilidade e evolução incremental do sistema. A separação entre scripts de coleta e camada de persistência (via DAL) permite fácil manutenção, inclusão de novas fontes e adaptação dinâmica dos parâmetros de busca pelo arquivo search.json.

5.1.4.2 Pré-processamento dos Dados

Após a coleta, os textos extraídos dos repositórios técnicos e as transcrições de vídeos do YouTube passam por uma etapa fundamental de pré-processamento com técnicas de Processamento de Linguagem Natural (PLN). O principal objetivo é normalizar, estruturar e enriquecer os dados textuais para viabilizar análises mais precisas e comparáveis nas etapas seguintes.

O pré-processamento é composto por diferentes etapas e módulos, descritos a seguir.

- **Limpeza e Normalização de Texto** — Utilizando o módulo Extractor.py, os textos coletados passam por um processo de limpeza com etapas variadas:
  - Remoção de caracteres especiais, URLs, emojis e repetições indesejadas, aplicando expressões regulares.
  - Conversão de HTML para texto puro (via html2text), eliminando qualquer marcação residual das páginas web.
  - Tokenização e Lematização: Com apoio da biblioteca spaCy, as frases são divididas em tokens (palavras), que por sua vez são reduzidas à sua forma base (lemas),

garantindo uniformidade e facilitando a comparação entre termos semanticamente equivalentes.

– Remoção de stopwords: Termos sem valor semântico, como artigos e preposições, são descartados para reduzir ruído.

O resultado é um JSON estruturado com o texto limpo, com idioma e as palavras-chave mais frequentes detectados, pronto para as próximas etapas do fluxo.

Exemplo prático:

Entrada original: "Nova técnica de desbloqueio FRP para o Moto G100 sem necessidade de PC."

Após o pré-processamento:

Texto limpo: "nova técnica desbloqueio frp moto g100 necessidade pc"

- **Transcrição de Vídeos** — O módulo transcript.py é responsável por extrair automaticamente as transcrições dos vídeos armazenados com source "youtube", utilizando a biblioteca youtube_transcript_api. As transcrições brutas são inicialmente salvas na tabela article, mantendo a rastreabilidade da fonte.

source_id	transcript	trar
youtube	Fala galera bem-vindo aqui ao canal do raposo estamos junto trazendo aqui um...	## [
youtube	Fala galera bem-vindo aqui ao canal do raposo diante de todos os obstáculos o...	## [
youtube	fala galerinha bem-vindo aqui ao canal do raposo mais um vídeo aqui atualizado...	## [
youtube	Fala galera bem-vindo aqui ao canal do raposo e vamos para o vídeo agora che...	## F
youtube	fala galerinha quanto tempo não vejo é mas se eu voltei aqui voltei por uma bo...	## [
youtube	fala galerinha bem-vindo aqui ao canal do raposo tamamo junto chegou essa p...	## [
youtube	vamos lá vamos lá estamos junto Raposo na voz estamos junto dos mais uma p...	## [

Figura 19 – Tabela do BD com as transcrições brutas armazenadas

- **Reestruturação de Transcrições com LLM** — Devido à informalidade e fragmentação das legendas extraídas do YouTube, aplica-se um processo adicional de reestruturação textual, realizado pelo módulo transcript_generative.py. Este módulo utiliza modelos generativos de linguagem natural, como ChatGPT (OpenAI) ou Gemini (Google), enviando as transcrições por meio de prompts pré-definidos e recebendo versões mais coesas, técnicas e organizadas em formato passo a passo. O conteúdo reestruturado é armazenado nos campos transcript_chatgpt ou transcript_gemini da tabela article, conforme o modelo utilizado.

transcript_gemini	transcript_chatgpt
## Desbloqueando o Google Play Service no Moto G54: Gui...	**Descrição Geral:** Neste tutorial, demonstr...
## Desbloqueando um Moto G7 Play com FRP: Guia Comple...	### Descrição Geral Neste tutorial, vamos de...
## Desbloqueando seu Moto G04/G04s/G4e com Android 1...	## Descrição Geral Neste vídeo, o canal apres...
## Resolvendo o problema de configurações bloqueadas n...	**Descrição Geral:** Este guia foi elaborado p...
## Desbloqueando seu Moto E13 sem Computador (Atualiz...	**Descrição Geral** O desbloqueio do Moto E1...
## Desbloqueando seu Moto G41 com Android 12 (Método ...	### Descrição Geral Neste tutorial, vamos det...
## Desbloqueando o FRP do Moto G32: Guia Completo ##...	**Descrição Geral:** Este guia explicará o pro...
## Desbloqueando seu Moto E32: Guia Completo Este guia...	### Descrição Geral Neste guia, você aprend...

Figura 20 – Tabela do BD com as transcrições reestruturadas

A aplicação automatizada do pré-processamento com uso de PLN assegura que todos os artigos — independentemente da origem — estejam normalizados, limpos e enriquecidos, tornando possível realizar análises comparativas de alta qualidade. A arquitetura permite que novas técnicas de pré-processamento ou modelos de linguagem sejam integrados de forma incremental, adaptando-se a requisitos futuros e à evolução do ecossistema Android.

Por fim, é importante ressaltar que, durante a validação da interface, avaliou-se a reestruturação de transcrições com dois modelos de linguagem natural: ChatGPT(OpenAI) e Gemini (Google). Embora ambas as soluções tenham sido demonstradas e discutidas com a equipe da Motorola, optou-se por utilizar o ChatGPT, devido à sua maior eficácia e melhor integração ao fluxo do sistema. A interface ainda permite a visualização das versões geradas pelo Gemini, mas o pipeline de análise de similaridade foi conduzido exclusivamente sobre as transcrições reestruturadas pelo ChatGPT, em conjunto com o SBERT. O uso do Gemini permanece como alternativa viável para evoluções futuras.

5.1.4.3 Mineração de Dados e Análise de Similaridade

A etapa de mineração de dados e análise de similaridade representa o núcleo analítico do sistema desenvolvido, sendo responsável pela identificação de padrões, recorrências e redundâncias em artigos técnicos (coletados do BypassFRPFiles.com) e artigos derivados de vídeos do YouTube. Este processo visa não apenas reconhecer conteúdos duplicados ou altamente relacionados, mas também contribuir para a priorização eficiente das vulnerabilidades que demandam atenção imediata.

- **Extração de Termos-Chave e Categorização Semiautomática:**

  Após o pré-processamento, os textos passam pelo módulo Extractor.py, responsável pela identificação de termos técnicos, expressões recorrentes e elementos relevantes para a

categorização dos artigos. O script utiliza expressões regulares e listas padronizadas, previamente armazenadas no banco de dados, para detectar menções a fabricantes, modelos e versões do Android, mesmo que apareçam de forma informal ou abreviada nos textos. Assim, cada artigo é automaticamente classificado quanto ao dispositivo e à versão do sistema operacional tratados, facilitando a filtragem posterior e a análise de tendências.

- **Avaliação de Similaridade entre Artigos:** O reconhecimento de conteúdos semelhantes é conduzido por métodos complementares, adaptados ao tipo e extensão dos textos:

  - **Distância de Levenshtein para Textos Curtos:** Títulos e descrições dos artigos são comparados usando o algoritmo de Levenshtein, que mede o número mínimo de operações necessárias para transformar uma string em outra. Essa abordagem é ideal para identificar duplicidades e pequenas variações em relatos curtos, como tutoriais técnicos ou resumos de vídeos. Os resultados desse cálculo são normalizados e registrados na tabela similarity, associando os pares de artigos comparados e o respectivo score.

article1_id	article2_id	similarity
30	31	55
30	32	60
30	33	78
30	34	38
30	35	46
30	36	40
30	37	8

Figura 21 – Tabela do BD similarity, armazenamento da similaridade Levenshtein com pares associados e score.

  - **Similaridade Semântica com SBERT para Textos Longos:** Conforme definido na etapa anterior, apenas as transcrições reestruturadas via ChatGPT foram utilizadas como insumo para o cálculo de similaridade semântica com SBERT. Para as transcrições reestruturadas (armazenadas nos campos transcript_chatgpt ou transcript_gemini), emprega-se o modelo "*paraphrase-multilingual-mpnet-base-v2*" do SBERT, via biblioteca sentence-transformers. Esse modelo, além de gerar embeddings vetoriais densos que capturam o significado semântico dos textos, pos-

sui suporte a múltiplos idiomas, fator essencial dada a natureza internacional do conteúdo extraído. A similaridade é calculada com base na distância de cosseno entre os embeddings dos artigos, sendo os resultados registrados na tabela similarity_transcript.

article1_id	article2_id	similarity
30	31	57
30	32	79
30	33	51
30	34	55
30	35	54
30	36	67
30	37	59
30	38	76

Figura 22 – Tabela do BD similarit_transcript, armazenamento da similaridade SBERT com pares associados e score.

É importante destacar que todos os pares de artigos são analisados, sem limiar fixo pré-estabelecido; assim, é possível detectar relações relevantes mesmo abaixo de valores convencionais de corte, com priorização dos três pares de maior similaridade exibidos na interface web para cada artigo.

- **Arquitetura e Fluxo do Módulo similarity.py:** O módulo similarity.py centraliza a lógica de comparação e armazenamento das similaridades. Seu funcionamento segue um fluxo estruturado:
  - **Gatilho:** A cada novo artigo inserido no banco (de qualquer fonte) ou atualização de transcrição processada por LLM, o sistema executa o módulo de similaridade.
  - **Extração:** Recupera do banco os textos relevantes para comparação (títulos, descrições ou transcrições reestruturadas).
  - **Cálculo:** Realiza as comparações por Levenshtein (para textos curtos) e por SBERT (para transcrições), armazenando os resultados nas respectivas tabelas.
  - **Registro:** Todos os pares comparados têm o score, o tipo de conteúdo e os IDs dos artigos envolvidos devidamente registrados, garantindo rastreabilidade e possibilidade de auditoria.

  - **Consulta:** A interface web consulta essas tabelas para exibir, de forma priorizada, os pares mais similares para cada artigo, facilitando a revisão pelos analistas de segurança.

- **Decisões Técnicas e Alinhamento com o Projeto**

  Essa abordagem híbrida — combinando análise lexical (Levenshtein) e análise semântica (SBERT) — permite que o sistema capture tanto duplicidades simples quanto relações profundas de conteúdo, mesmo quando a linguagem varia significativamente. O uso do modelo **paraphrase-multilingual-mpnet-base-v2** garante robustez diante de diferentes idiomas e estilos textuais, o que é especialmente relevante em um cenário globalizado de relatos de vulnerabilidade. O registro completo dos pares permite análises históricas e facilita auditorias, enquanto a priorização dinâmica dos pares mais similares contribui para uma triagem ágil dos casos críticos.

5.1.4.4 Armazenamento e Organização

O armazenamento e a organização dos dados no sistema constituem uma etapa fundamental para garantir a integridade, rastreabilidade e eficiência na análise dos artigos coletados. Para isso, foi adotada uma arquitetura baseada em banco de dados relacional (MySQL), complementada por uma camada intermediária de abstração — a **Data Access Layer (DAL)** — responsável por encapsular todas as operações de persistência, consulta e atualização das informações.

5.1.4.4.1 Estrutura do Banco de Dados

O modelo de dados foi desenhado segundo boas práticas de normalização, com o objetivo de minimizar redundâncias e preservar a integridade referencial. As principais entidades modeladas no banco incluem:

- **article:** Tabela central para armazenamento dos artigos coletados, contendo campos para título, descrição, data de publicação, link, origem (site técnico ou vídeo), e campos específicos para transcrições brutas e processadas por modelos de linguagem.
- **vendor** e **model:** Estruturas para catalogação de fabricantes e modelos de dispositivos, permitindo associação precisa dos artigos aos equipamentos discutidos.

- **android_version:** Registra os diferentes releases do sistema operacional Android, possibilitando o rastreio de tendências de vulnerabilidades por versão.
- **Tabelas de associação** (article_model, article_android_version): Viabilizam o relacionamento n:N entre artigos, modelos de dispositivo e versões do Android, conforme identificado durante o processamento textual automatizado.
- **similarity** e **similarity_transcript:** Armazenam os resultados das análises de similaridade realizadas, associando pares de artigos com elevado grau de correspondência lexical ou semântica.
- **articleview:** View consolidada que une dados de artigos, modelos, fabricantes e versões do Android, otimizando consultas para visualização na interface web.

A criação e a manutenção do esquema relacional são automatizadas pelo script database_creation.py, reduzindo erros humanos e padronizando atualizações do banco ao longo do ciclo de vida do projeto.

5.1.4.4.2 Camada DAL e Classes de Persistência

A interação com o banco é inteiramente mediada pelas classes da camada DAL, localizadas na pasta /database. Esse padrão desacopla a lógica de negócio dos detalhes de persistência, favorecendo manutenção, testes e evolução futura do sistema. Entre as principais classes, destacam-se:

- **GenericDAL:** Classe base com métodos genéricos de CRUD e tratamento de exceções, herdada pelas demais classes DAL.
- **ArticleDAL:** Responsável por inserir e atualizar artigos, manipular transcrições, identificar duplicidades e associar modelos e versões do Android aos artigos.
- **ModelDAL** e **VendorDAL:** Gerenciam respectivamente as tabelas de modelos e fabricantes.
- **SimilarityDAL:** Registra os resultados das comparações por Levenshtein e SBERT, centralizando as consultas de similaridade exibidas na interface.
- **ArticleViewDAL:** Permite consulta otimizada e agregada à view articleview, filtrando e organizando os dados para o frontend.

Todo acesso ao banco — seja oriundo dos scripts de coleta, dos processamentos intermediários ou da interface web (via app.py) — ocorre exclusivamente por meio dessas classes, reforçando a integridade dos dados e a rastreabilidade das operações.

5.1.4.4.3 Decisões Técnicas e Alinhamento com a Usabilidade

A separação entre lógica de persistência, negócio e apresentação garante não apenas robustez, mas também flexibilidade para futuras expansões do sistema. A presença da view consolidada articleview simplifica e acelera o carregamento das informações na interface web, tornando as análises dos especialistas mais ágeis e confiáveis. A adoção rigorosa de boas práticas de modelagem e desenvolvimento, refletidas na organização das classes DAL, permite que o sistema evolua conforme surgem novas demandas ou fontes de dados.

A arquitetura aqui detalhada está completamente alinhada com as diretrizes técnicas apresentadas no Capítulo 4 e com o fluxo de código do projeto, viabilizando tanto consultas eficientes quanto a geração de relatórios analíticos a partir dos dados coletados.

Essa organização é essencial para consultas eficientes e geração de relatórios, além de facilitar a integração com a interface web descrita na seção seguinte (5.1.4.5).

5.1.4.5 Interface Web para Análise e Visualização

A interface web é a principal ponte entre o processamento automatizado de dados e a atuação analítica dos profissionais de segurança, consolidando em um ambiente único os resultados das coletas, análises e comparações realizadas pelo sistema. Desenvolvida com o framework Flask, a aplicação oferece recursos interativos de visualização, filtragem, marcação, comentários e exportação de dados, adaptando-se às necessidades específicas do fluxo de análise de vulnerabilidades.

5.1.4.5.1 Arquitetura e Integração Backend-Frontend

A aplicação Web foi desenvolvida com base na arquitetura cliente-servidor. O backend, implementado em Python pelo arquivo app.py, gerencia todas as rotas, autenticação OAuth, lógica de negócio e integração com as classes da camada DAL. O frontend, construído sobre HTML, CSS e JavaScript, utiliza templates Jinja2 para renderização dinâmica dos dados provenientes da view consolidada articleview do MySQL.

A segurança é reforçada pelo uso de autenticação OAuth via Google, restrita ao domínio @motorola.com, assegurando que apenas colaboradores autorizados tenham acesso à plataforma e seus dados sensíveis.

5.1.4.5.2 Fluxos Específicos das Telas

A interface foi segmentada em módulos funcionais, cada um adaptado à natureza da fonte de dados e aos requisitos de usabilidade, conforme os códigos e templates enviados:

- **Tela do YouTube (/youtube)**
  - **Tabela Paginada de Artigos:** Exibe os artigos extraídos de vídeos do YouTube, apresentando colunas como título, fabricante, modelo, versão Android, data, origem e índices de similaridade (Levenshtein e SBERT).
  - **Filtros Dinâmicos:** O analista pode filtrar artigos por fabricante, modelo, versão do Android, além de ordenar por data ou score de similaridade, promovendo um refinamento eficiente da análise.
  - **Busca por ID:** Campo exclusivo para localização direta de um artigo específico a partir de seu identificador único (ID), agilizando revisitas a registros previamente analisados.
  - **Marcação de Visualização:** Após análise, o artigo pode ser marcado como "visualizado", sendo automaticamente transferido para uma aba dedicada e ocultado da lista principal, o que apoia o controle do progresso da triagem.
  - **Exportação CSV:** Possibilita exportar os artigos filtrados para planilhas em formato CSV, facilitando análises externas e relatórios corporativos.
  - **Comentários Técnicos:** Os analistas podem inserir observações e comentários diretamente nos artigos, promovendo um ambiente colaborativo e registrando decisões.
  - **Visualização de Transcrições:** Botões específicos permitem acessar as transcrições originais e as versões reestruturadas por modelos LLM (ChatGPT e Gemini), retornando os dados via requisições assíncronas (JSON).
  - **Comunicação Assíncrona:** Operações como marcação de visualização, adição de comentários e acesso às transcrições são processadas em segundo plano via chamadas AJAX (JSON), garantindo responsividade e fluidez.

- **Tela do BypassFRPFiles (/bypassfrpfiles)**

  - **Tabela de Artigos Coletados:** Lista artigos provenientes do site técnico BypassFRPFiles.com, com campos similares à tela do YouTube, porém sem transcrições.
  - **Filtros por Fonte/Data:** Os artigos exibidos já estão filtrados pela origem (bypassfrpfiles), podendo ser organizados por data de coleta.
  - **Paginação:** Navegação eficiente entre páginas, suportada pelo template Jinja2. **Exportação CSV:** Exportação dedicada dos artigos dessa fonte para CSV.
  - **Comentários Técnicos:** Registro e consulta de comentários sobre cada artigo, promovendo documentação colaborativa.
  - **Comunicação Tradicional e Assíncrona:** As principais operações utilizam rotas Flask/Jinja2, com ações colaborativas (comentários) podendo utilizar chamadas JSON.

- **Tela Inicial e Modularidade**

  Após autenticação, a tela inicial apresenta botões que disparam, sob demanda, os scripts de coleta:

  - **Models Scraping:** Atualização dos fabricantes/modelos pelo script smartphones_models_scraping.py.
  - **bypassfrpfiles.com:** Coleta manual de tutoriais técnicos, via webscraping_monograph.py.
  - **YouTube:** Busca, transcrição e análise dos vídeos, acessando o módulo YouTube.

Em alinhamento com o Capítulo 4, essa organização modular permite à equipe de segurança maior controle sobre as entradas do sistema, promovendo coleta dirigida e validação incremental das fontes.

A interface foi estruturada para manter a rastreabilidade das operações (marcação de visualizado, comentários, histórico de exportações), centralizando em um único ambiente todas as ações necessárias ao fluxo de análise. O uso da view articleview no backend garante consultas otimizadas, enquanto a divisão de funcionalidades entre as rotas e templates (youtube_listed.html, bypassfrpfiles_listed.html) espelha a arquitetura lógica dos módulos de coleta, processamento e persistência.

A interface web representa um avanço significativo frente aos métodos manuais anteriores, integrando automação, usabilidade e colaboração em um ambiente seguro. Ao viabilizar a triagem eficiente, o registro de histórico e a exportação flexível dos dados, o sistema potencializa a produtividade e a assertividade dos analistas, apoiando a identificação proativa de vulnerabilidades críticas no ecossistema Android.

As figuras 23 até 34 a seguir trazem imagens de uso da interface.

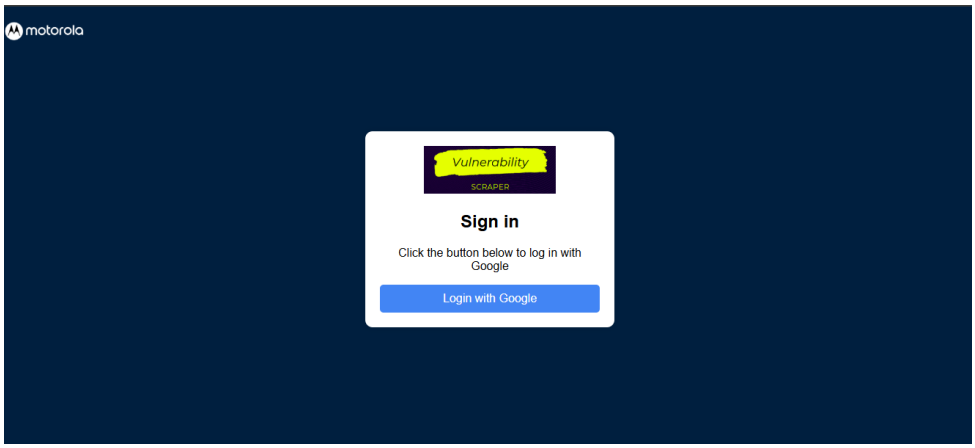

Figura 23 – Tela de autenticação de login e senha restrita com autenticação do Google usando conta Motorola. Fonte: Autor.

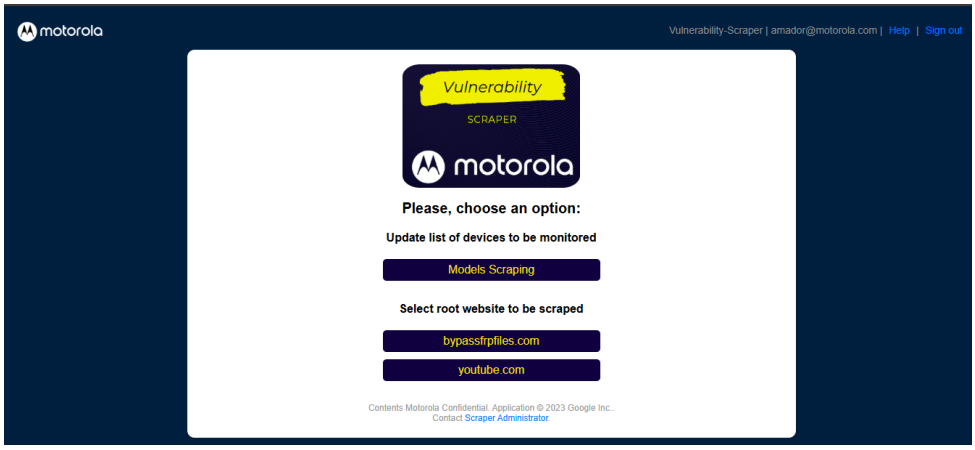

Figura 24 – Tela inicial da interface após login com autenticação OAuth. Fonte: Autor.

Embora esta subseção apresente majoritariamente as telas da interface web, as Figuras 33 e 34 foram incluídas para ilustrar o funcionamento interno do módulo de análise de similaridade. Os resultados exibidos ainda não estão integrados visualmente à interface, mas são gerados pelo backend do sistema e armazenados no banco de dados para posterior visualização. Essas saídas exemplificam o cálculo realizado pelas métricas Levenshtein e SBERT, permitindo validar a consistência e a correlação entre ambas.

Figura 25 – Atualização de novos modelos dos principais fabricantes. Fonte: Autor.

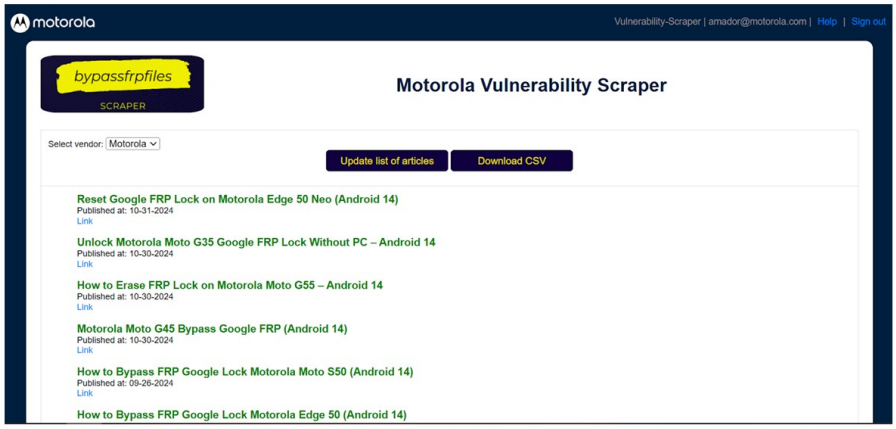

Figura 26 – Extração de artigos do website "FRP Bypass"com marcação de metadados referentes a fabricante, modelo, versão do Android e conteúdo textual para análise. Fonte: Autor.

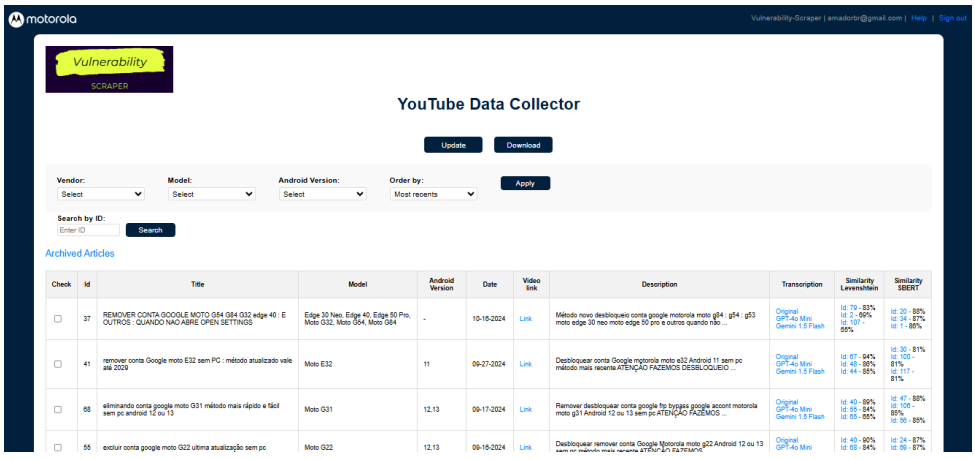

Figura 27 – Tela de filtros para configurar o modo da busca que será realizada. Fonte: Autor.

☐	45	DESBLOQUEIO CONTA GOOGLE MOTO G04 ANDROID 14 SEM PC	Moto G04	14	03-28-2024	Link	como remover conta Google Motorola moto g04 Android 14 sem pc Quem não conseguir veja esse vídeo ...	View GPT-4o Mini Gemini 1.5 Flash	Id: 1 - 91% Id: 36 - 72% Id: 4 - 69%	Id: 101 - 89% Id: 92 - 89% Id: 118 - 88%
☐	30	SAIUUU ! REMOVER CONTA GOOGLE MOTO E22 SEM PC MÉTODO INFALÍVEL ASSISTA O VÍDEO 2 MÉTODOS	Moto E22	12	03-24-2024	Link	como desbloquear conta google motorola moto e22 sem pc 2 método veja o vídeo sem pc android 12 quem não conseguiu como ...	View GPT-4o Mini Gemini 1.5 Flash	Id: 15 - 95% Id: 69 - 94% Id: 100 - 94%	Id: 44 - 82% Id: 41 - 81% Id: 32 - 79%
☐	3	SAIUU !!! REMOVER CONTA GOOGLE MOTO G30 MOTO G31 ANDROID 12 SEM PC MÉTODO NOVÍSSIMO	Moto G30, Moto G31	12	03-23-2024	Link	como remover conta Google Motorola moto g31 moto g31 Android 12 sem pc novo método ATENÇÃO FAZEMOS DESBLOQUEIO ...	View GPT-4o Mini Gemini 1.5 Flash	Id: 21 - 75% Id: 85 - 75% Id: 47 - 73%	Id: 4 - 86% Id: 20 - 86% Id: 1 - 85%
☐	49	REMOVER CONTA GOOGLE MOTO G60 G60S VALE ATÉ 2029 ATUALIZADO SEM USAR O YOUTUBE	Moto G22, Moto G60, Moto G60S	12,13	03-21-2024	Link	desbloqueio conta Google Motorola Android 12 ou Android 13 sem pc moto g60 g60s g22 ete... método novo vale até 2029 ...	View GPT-4o Mini Gemini 1.5 Flash	Id: 55 - 67% Id: 40 - 66% Id: 88 - 66%	Id: 82 - 89% Id: 40 - 88% Id: 101 - 88%
☐	54	como desbloquear conta Google moto G8 Power lite Android 10 ou 11 sem pc	Moto G8 Power Lite	10,11	03-12-2024	Link	como remover conta Google Motorola moto g8 Power lite sem muito detalhes sem pc ATENÇÃO FAZEMOS DESBLOQUEIO DE ...	View GPT-4o Mini Gemini 1.5 Flash	Id: 91 - 71% Id: 17 - 69% Id: 64 - 69%	Id: 64 - 80% Id: 76 - 80% Id: 46 - 79%
☐	63	DESTRUIR CONTA GOOGLE MOTO G20 E G10 sem pc - método novíssimo : sem detalhes	Moto G10, Moto G20	-	03-11-2024	Link	como desbloquear remover conta Google Motorola moto g10 e moto g20 sem pc e sem usar configuração do Android método ...	View GPT-4o Mini Gemini 1.5 Flash	Id: 74 - 70% Id: 84 - 70% Id: 38 - 68%	Id: 79 - 91% Id: 83 - 90% Id: 90 - 87%

1 2 3 4 5 6 Next

Figura 28 – Tela de visualização de novos artigos (vídeos) do Youtube sobre FRP, apresentando o dispositivo afetado e cálculos de similaridade com relação a artigos já existentes no BD local. Fonte: Autor.

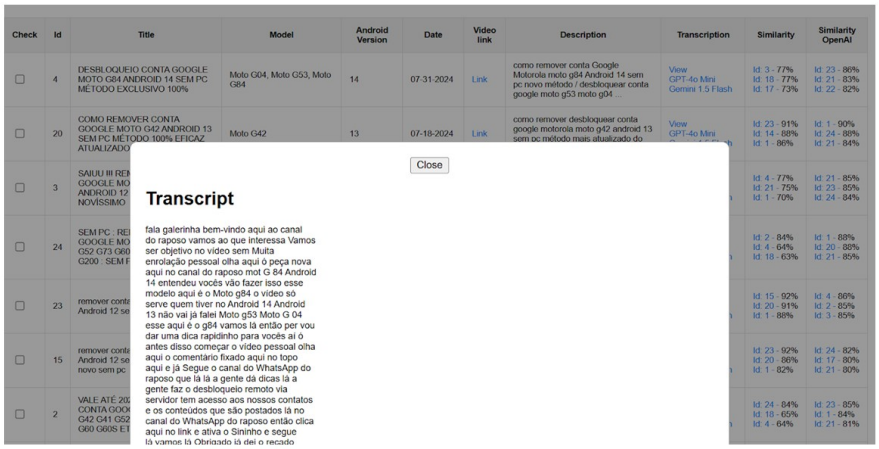

Figura 29 – Visualização das transcrições originais de áudio. Fonte: Autor.

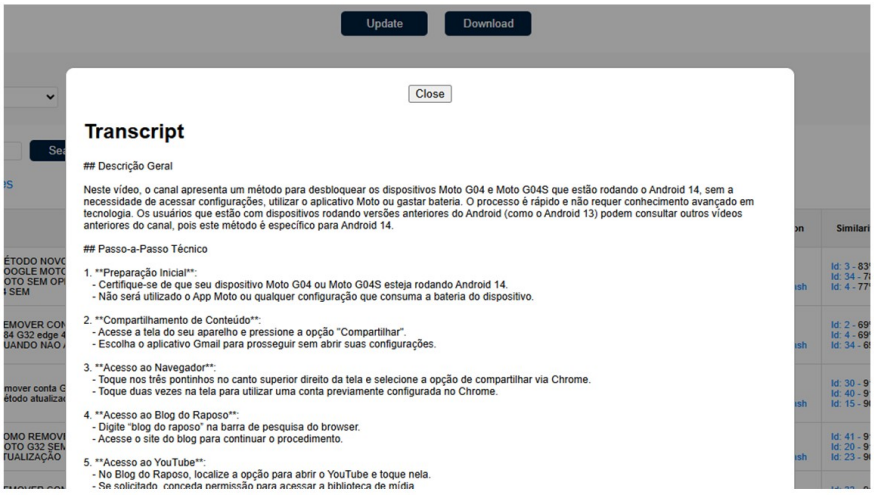

Figura 30 – Tela de visualização das transcrições após pré-processamento com LLM: Descrição Geral + Passo-a-passo técnico. Fonte: Autor.

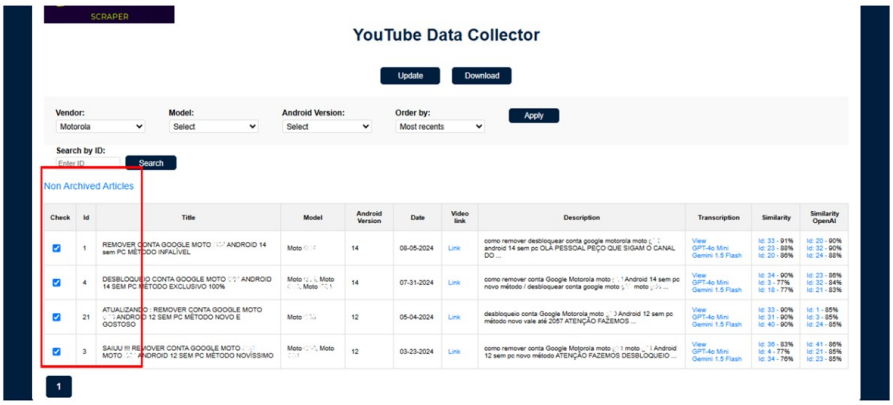

Figura 31 – Funcionalidade para arquivamento de artigos que foram pesquisados e analisados. Fonte: Autor.

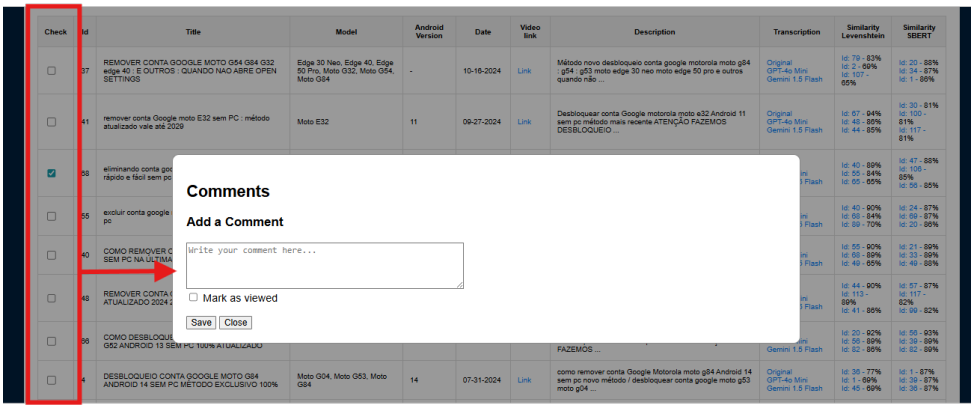

Figura 32 – Tela de verificação, permitindo adicionar comentários e marcar artigo como visualizado. Fonte: Autor.

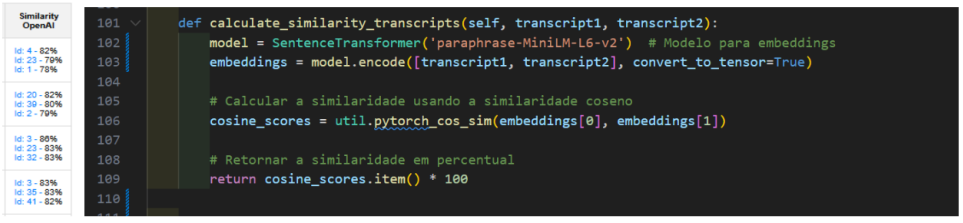

Figura 33 – Cálculo da similaridade usando a biblioteca *Sentence Transformer* (SBERT). Fonte: Autor.

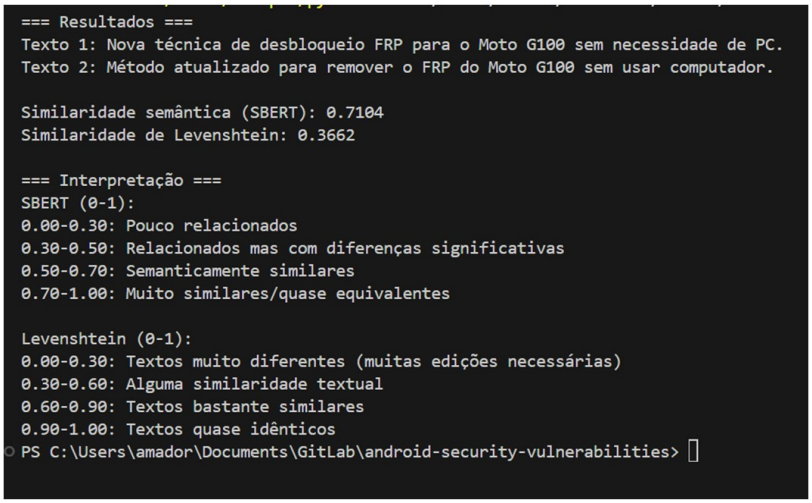

Figura 34 – Exemplo de valores de similaridade usando a medida de Levenshtein e o SBERT. Fonte: Autor.

5.2 TESTES E RESULTADOS

Após a conclusão da implementação do protótipo, foram realizados testes para validar sua eficácia na coleta, organização e análise de relatos sobre vulnerabilidades no Android. Como já mencionado, o foco inicial foi na vulnerabilidade *Factory Reset Protection*. Essa etapa teve como objetivo verificar se o sistema era capaz de executar, de forma automatizada e confiável, todas as funcionalidades planejadas, reduzindo o esforço manual anteriormente necessário.

Além disso, buscou-se avaliar o desempenho dos métodos para cálculo de similaridade e, por fim, a usabilidade da interface desenvolvida para uso pelos especialistas em segurança. A seguir, são descritos a metodologia adotada para os testes e os principais resultados obtidos.

5.2.1 Metodologia de Testes

A avaliação do protótipo baseou-se em cenários reais de análise de vulnerabilidades, valendo-se de dados extraídos de fontes públicas como o site BypassFRPFiles e vídeos do YouTube. O fluxo completo do sistema foi executado com os dados, contemplando desde a coleta automatizada até a visualização dos resultados na interface web.

Etapas de teste:

- Coleta de dados: Foram coletados 24 artigos a partir do site FRP Bypass, e 103 transcrições de vídeos do YouTube, a partir dos canais já mencionados acima. A coleta incluiu diferentes modelos de dispositivos Android e múltiplas versões desse SO, abrangendo assim um espectro representativo dos relatos encontrados na prática.
  - Vale salientar que 2 dos vídeos coletados não traziam a transcrição para texto, mas foram mantidos na base de teste porque o cálculo referente à sua parte textual é realizado de modo independente, sendo apresentado via interface do usuário. Como mencionado, esses vídeos não são descartados porque a informação de similaridade parcial ainda é melhor do que nenhuma informação sobre o vídeo.
- Pré-processamento: os dados coletados foram pré-processados de acordo com sua origem, como definido na Seção 4.3 do Capítulo 4. A seção 4.3.1 apresenta as etapas de pré-processamento dos dados oriundos do site FRP Bypass e da parte textual do vídeos,

resultando em uma lista de tokens. Já a Seção 4.3.2 traz as etapas de tratamento das transcrições de vídeos do YouTube, resultando em um texto mais completo.

- Mineração de dados: A seguir, as representações dos dados obtidas do pré-processamento passaram pela etapa de mineração, como descrito na Seção 4.4. O cálculo de similaridade é a fase de maior interesse desta etapa, tendo sido também realizado de modo diferente, a depender da origem do artigo. Os dados oriundos do site FRP Bypass são comparados entre si através da medida de Levenshtein. Os dados oriundos da parte textual do vídeos também são comparados através da medida de Levenshtein. Contudo, é importante salientar que dados de origens diferentes não são comparados entre si, por serem de natureza distinta. Assim, os artigos do FRP Bypass não serão comparados com a parte textual dos artigos oriundos de vídeos do YouTube. De fato, esses dados são armazenados no BD separadamente. Por fim, as transcrições de vídeos do YouTube são comparadas com auxílio do SBERT, por se tratarem de textos mais extensos. Aqui também havia a intenção de capturar a semântica dos textos, que não é tratada pela medida de Levenshtein.

  - Observação: quando o sistema está em uso diário, os dados novos também são comparados com os que já estão registrados no Banco de dados. Apenas nessa fase de testes não existia ainda nada guardado no BD.

- Armazenamento: Os dados já processados são etiquetados e armazenados no BD MySQL juntamente com as taxas de similaridade calculadas anteriormente.

- Visualização: Por fim, todas essas informações ficam disponíveis para visualização através da interface de consulta.

- Análise dos resultados: Por fim, os dados armazenados foram examinados, a fim de se calcular a métrica de precisão da etapa de mineração. Adicionalmente, foram realizadas análises qualitativas com esses dados.

Durante os testes, também foram observados indicadores de tempo médio de processamento, com o objetivo de constatar a grande contribuição desse processo semiautomático, quando comparado ao modo manual de coleta e análise realizado pela empresa parceira. Esses testes deram uma ideia da enorme redução do esforço humano, com ganhos em velocidade e consistência dos dados coletados, além de apontar oportunidades para refino do sistema.

5.2.2 Resultados Obtidos

Como mencionado acima, a avaliação dos resultados do sistema protótipo considerou diferentes abordagens, iniciando por uma análise quantitativa baseada na métrica tradicional de Precisão Seção 5.2.2.1, complementada por análises visuais qualitativas Seção 5.2.2.2, que auxiliaram no melhor entendimento dos resultados obtidos com as métricas de similaridade entre os artigos.

5.2.2.1 Avaliação Quantitativa com Métrica de Precisão

Essa fase dos testes teve como objetivo analisar a precisão das taxas de similaridade calculadas automaticamente. Como tratamos dois tipos distintos de artigos (FRP Bypass e vídeos), esse teste foi replicado para os dois conjuntos de dados.

Inicialmente, os artigos de cada conjunto foram combinados em pares, uma vez que o objetivo é avaliar a similaridade entre pares de artigos. O conjunto oriundo do FRP Bypass, com 24 artigos, deu origem a 276 pares. O conjunto dos vídeos, com 103 artigos, deu origem a 5253 pares referentes à sua parte textual (título e descrição). Porém, como 2 dos vídeos coletados não traziam a transcrição para texto, foram obtidos 5050 pares com a similaridade realizada pelo SBERT.

A partir desses conjuntos, foram gerados rankings separados, ordenados pela taxa de similaridade entre pares. A seguir, foi conduzida uma análise manual dos 100 pares no topo de cada ranking. Não foi possível analisar a totalidade dos pares por restrições de tempo. A marcação manual foi binária, indicando apenas se, no julgamento humano, os pares analisados são de fato similares no tocante à vulnerabilidade relatada. Marcamos os relatos relevantes com "1" e os não relevantes com "0", de acordo com seu conteúdo técnico.

A Figura 35 ilustra os 10 pares no topo do ranking obtido pela comparação dos dados oriundos dos vídeos. Escolhemos mostrar esse ranking porque as análises realizadas para as comparações via Levenstein deram 100% de acerto.

Nota-se uma discordância logo na posição 4 do ranking. Nesse caso, as duas transcrições são quase idênticas, contudo diferem no final do relato no tocante à vulnerabilidade sendo explorada. Assim, de acordo com o julgamento humano, esses artigos não são similares.

A seguir, a precisão do cálculo de similaridade com SBERT foi estimada a partir dessa marcação manual dos 100 artigos melhor ranqueados. Foi aplicada a métrica de precisão

Ranking	A1	A2	Similaridade SBERT	Avaliação Manual
1	56	66	93	1
2	50	80	92	1
3	20	24	91	1
4	24	112	91	0
5	56	82	91	1
6	63	79	91	1
7	20	34	90	1
8	63	83	90	1
9	79	83	90	0
10	92	101	90	1

Figura 35 – Ranking dos 10 primeiros pares no Top-100 — similaridade entre transcrições calculada com SBERT. Fonte: Autor.

(Seção 2.2.5.1) como método de avaliação da efetividade do sistema na priorização de relatos relevantes.

A partir desse processo, foram identificados:

- Verdadeiros Positivos (VP) = 55 artigos similares identificados corretamente nos Top-100
- Falsos Positivos (FP) = 45 artigos não similares incluídos nos Top-100

Com esses valores, foi possível calcular a **precisão** do sistema para os Top-100 pares do ranking por meio da fórmula clássica:

$$\text{Precisão} = \frac{VP}{VP + FP}$$

Substituindo-se os valores encontrados pela aálise manual, temos que:

$$\text{Precisão} = \frac{55}{55 + 45}$$

Assim, a precisão Top-100 resultou em uma taxa de 55% de precisão. Ou seja, o sistema acertou no cálculo de similaridade para 55% dos pares priorizados no Top-100, demonstrando uma boa performance do sistema na triagem inicial de vulnerabilidades. Relembramos que a interface apresenta apenas 3 artigos similares por cada artigo sendo visualizado.

Além da precisão Top-100, foi também calculada a precisão acumulada para cada posição k do ranking (de 1 a 100), utilizando a fórmula a seguir:

$$\text{Precisão@}k = \frac{\text{num acertos até posição } k}{k}$$

Esse cálculo progressivo permitiu visualizar a evolução da precisão ao longo do ranking. A Figura 36 apresenta a curva gerada, que demonstra que o sistema alcançou valores próximos de 1.0 nas primeiras posições, mantendo-se acima de 80% até a 60ª posição e estabilizando-se em torno de 70% nas posições finais.

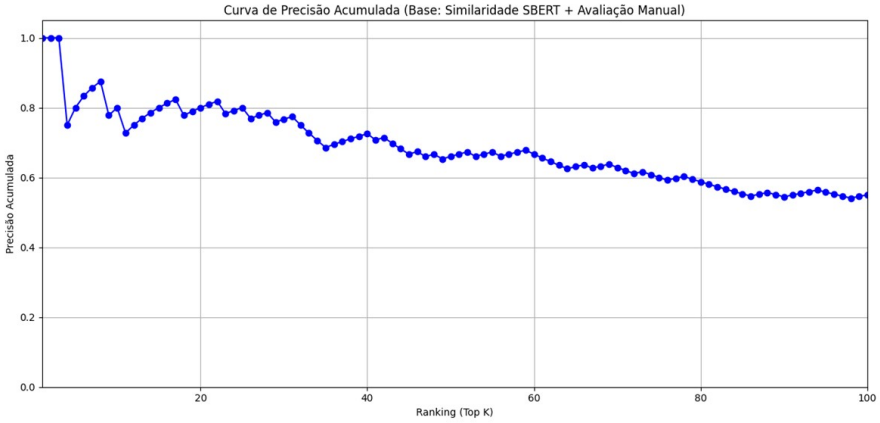

Figura 36 – Curva de Precisão Acumulada baseada na similaridade SBERT com avaliação manual binária. Fonte: Autor.

Esses resultados quantitativos demonstram que o sistema tem alta capacidade de priorizar relatos relevantes de forma semi-automatizada, o que é essencial em um contexto de segurança da informação, onde o tempo de resposta é crítico.

Por fim, vale salientar que não foi possível calcular a medida "revocação" (recall) do sistema, uma vez que não houve tempo hábil para avaliarmos a totalidade de pares de artigos gerada. Em consequência, também não foi possível calcular o F1-score, que depende do valor da revocação.

5.2.2.2 Análises Visuais e Qualitativas

Para complementar a avaliação numérica, também foram aplicadas técnicas visuais para facilitar a interpretação dos dados obtidos com o conjunto de transcrições de vídeos. As principais técnicas utilizadas foram (Seção 2.2.5.2):

- Mapas de Calor (Heatmaps): destacam os pares de relatos com maior grau de similaridade.
- Nuvens de Palavras (Wordclouds): revelam os termos mais frequentes nos relatos, após o pré-processamento.
- Gráficos de Dispersão: baseados em embeddings SBERT, ajudam a visualizar agrupamentos de relatos semanticamente próximos.

Essas representações foram geradas com base em um subconjunto de seis artigos (49, 56, 66, 82, 101 e 118), previamente selecionados por sua relevância. Os resultados confirmaram visualmente os agrupamentos coerentes gerados pelo sistema, reforçando a confiabilidade das técnicas empregadas.

5.2.2.2.1 Mapa de Calor (Heatmaps)

O mapa de calor (*heatmap*) foi utilizado para representar visualmente os níveis de similaridade semântica entre os seis artigos escolhidos, com base nos embeddings gerados pelo modelo SBERT. Cada célula da matriz indica o grau de semelhança entre dois textos, variando de 65 (menor similaridade) a 100 (máxima similaridade) — ver Figura 37.

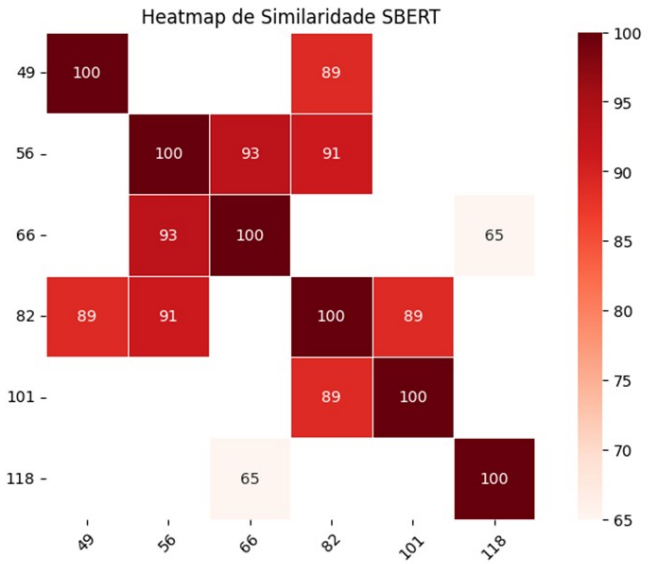

Figura 37 – Heatmap de Similaridade com SBERT entre os artigos 49, 56, 66, 82, 101 e 118. Fonte: Autor.

A partir da Figura 37, pode-se observar que os artigos 56, 66, 82 e 101 apresentam forte correlação semântica, formando um grupo coeso de relatos com vocabulário e conteúdo semelhantes. Em contrapartida, os artigos 49 e 118 destacam-se por valores de similaridade mais baixos em relação aos demais, sugerindo maior distância temática, possivelmente por abordarem variações de método, vocabulário técnico distinto ou subtemas dentro do contexto de desbloqueio FRP.

5.2.2.2.2 Nuvens de Palavras (Word Clouds)

As nuvens de palavras foram geradas individualmente para cada um dos seis artigos selecionados, com o objetivo de destacar os termos mais frequentes em cada relato. Essa técnica permite identificar rapidamente os tópicos centrais abordados nos textos (ver Figura 38).

Figura 38 – Nuvens de Palavras geradas para os artigos 49, 56, 66, 82, 101 e 118. Fonte: Autor.

Os termos mais proeminentes, como "configurações", "google", "procedimento", "aparelho" e "conta", aparecem com alta recorrência, revelando a centralidade do tema de desbloqueio FRP em dispositivos Motorola. Ao comparar as nuvens, nota-se uma variação na terminologia e no foco: enquanto alguns relatos enfatizam comandos e interfaces (*talkback*, comando de voz, assistente), outros se concentram em aspectos técnicos como instalação, versão ou navegador.

Essas visualizações permitem ao analista explorar rapidamente padrões de vocabulário, identificar termos emergentes, e diferenciar conteúdos recorrentes de relatos fora do padrão.

5.2.2.2.3 Gráficos de Dispersão (PCA dos Embeddings SBERT)

Para representar graficamente a distribuição semântica entre os artigos, foi utilizado o algoritmo PCA (*Principal Component Analysis*) aplicado sobre os *embeddings* gerados pelo modelo SBERT. Essa técnica reduz os vetores de alta dimensionalidade para duas dimensões, permitindo uma visualização clara da proximidade semântica entre os textos analisados (ver Figura 39).

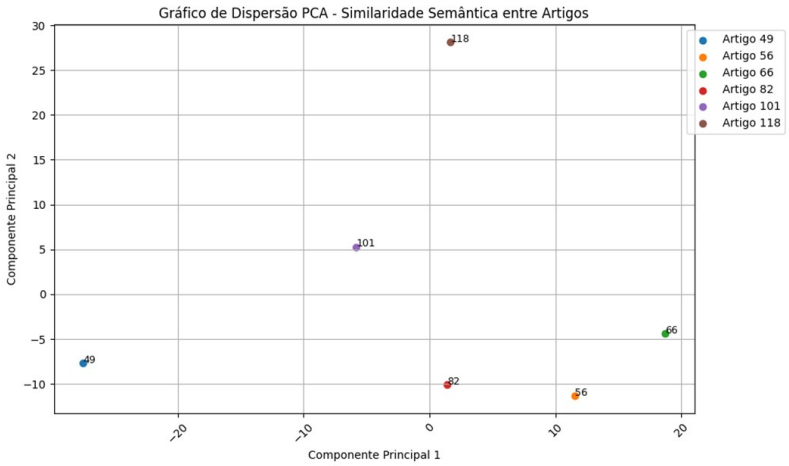

Figura 39 – Gráfico de Dispersão PCA dos artigos 49, 56, 66, 82, 101 e 118, evidenciando agrupamentos e distanciamentos semânticos com base no SBERT — os . Fonte: Autor.

Esse gráfico evidencia que os artigos 56, 66, 82 estão relativamente próximos entre si, indicando afinidade temática e similaridade de conteúdo. O artigo 101 ocupa uma posição intermediária, com proximidade moderada em relação ao grupo central, sugerindo alguma relação semântica, mas com características próprias. Por outro lado, os artigos 49 e 118 se destacam por estarem mais afastados dos demais, reforçando sua singularidade semântica — possivelmente devido a diferenças de abordagem, foco em subtemas distintos ou vocabulário específico.

Esse tipo de visualização é especialmente útil em análises exploratórias de dados, pois facilita a identificação de agrupamentos naturais, outliers e relações semânticas relevantes entre documentos. Além disso, pode subsidiar aplicações como classificação automática de

relatos, recomendação de conteúdos e comparação entre diferentes técnicas ou vulnerabilidades reportadas.

5.2.2.2.4 Síntese das Técnicas de Visualização

A aplicação combinada dessas três técnicas — *heatmap*, nuvens de palavras e gráfico de dispersão — reforça a eficácia do sistema na análise textual automatizada, destacando padrões, similaridades e diferenças entre os relatos coletados. Essa abordagem visual facilita a priorização de conteúdos relevantes e subsidia a tomada de decisão por parte de analistas de segurança, especialmente em contextos com grande volume de dados e múltiplos relatos sobre desbloqueio FRP em dispositivos Android.

5.3 CONSIDERAÇÕES FINAIS

O capítulo apresentou a implementação, os testes e os principais resultados do sistema protótipo desenvolvido para identificação e análise automatizada de vulnerabilidades em dispositivos Android, com foco inicial em técnicas de FRP (*Factory Reset Protection*). Os experimentos realizados demonstraram a viabilidade e os benefícios de uma abordagem automatizada e integrada, especialmente em cenários que exigem monitoramento constante de grandes volumes de informação.

A adoção de uma arquitetura modular e escalável permitiu a integração eficiente de múltiplos módulos de coleta, processamento e análise, mantendo a flexibilidade para evolução futura da solução. O uso de técnicas avançadas de Processamento de Linguagem Natural, como SBERT e LLMs, foi fundamental para tratar, padronizar e comparar textos extraídos de fontes heterogêneas, contribuindo para uma triagem mais precisa e relevante das vulnerabilidades identificadas.

A interface web implementada se mostrou eficiente para o uso prático por analistas, possibilitando filtros, comentários, marcações e exportações, além de proporcionar um controle rigoroso de acesso por meio de autenticação restrita.

Os resultados quantitativos e qualitativos indicam que o sistema é capaz de reduzir significativamente o esforço manual, aumentar a precisão na identificação de vulnerabilidades e agilizar o tempo de resposta da equipe de segurança. Além disso, a estrutura do sistema

facilita sua expansão para outros tipos de vulnerabilidades e fontes de dados, tornando-o uma base promissora para futuras pesquisas e aplicações em cibersegurança móvel.

Como perspectivas de continuidade, recomenda-se a inclusão de novos módulos para outras classes de vulnerabilidades, o aprimoramento dos algoritmos de análise semântica, e a ampliação da integração com plataformas de resposta a incidentes, de modo a evoluir para um sistema de monitoramento proativo e automático de ameaças.

6 CONCLUSÃO E TRABALHOS FUTUROS

A crescente exposição de vulnerabilidades em dispositivos móveis, especialmente via relatos publicados na web, motivou o desenvolvimento deste trabalho, que teve como objetivo propor uma abordagem semiautomatizada para identificação e extração dessas informações em dispositivos Android, com ênfase inicial na funcionalidade *Factory Reset Protection* (FRP).

Este capítulo apresenta uma síntese dos resultados alcançados, com foco nas respostas às questões de pesquisa, nas principais contribuições e perspectivas para trabalhos futuros.

6.1 RESPOSTAS ÀS QUESTÕES DE PESQUISA

Questão de Pesquisa 1: *Quais técnicas de IA, PLN e Web scraping são úteis e eficazes na extração e no processamento de dados relevantes sobre vulnerabilidades descritas em formato textual?*

Ao longo deste trabalho, foram investigadas, implementadas e avaliadas diversas técnicas das áreas de IA, PLN e Web scraping para extração e processamento de dados abertos em sites Web sobre vulnerabilidades em dispositivos Android.

Esses estudos foram determinantes para guiar as escolhas relacionadas ao processo proposto para identificação e extração de vulnerabilidades (Capítulo 4), bem como a construção do sistema protótipo (Capítulo 5). Como visto, o processo proposto seguiu as diretrizes de boas práticas da Engenharia de Software, adotando uma arquitetura modular e extensível. Cada módulo foi projetado para ser eficiente e fácil de modificar.

O sistema protótipo adotou tecnologias atuais e adequadas para tratar os problemas computacionais desta pesquisa, utilizando técnicas robustas de web scraping e APIs públicas para construir os módulos de coleta automática de textos provenientes de sites especializados, fóruns e vídeos do YouTube. O pré-processamento e a análise dos textos foram realizados com apoio de modelos avançados de PLN, incluindo o uso de LLMs como ChatGPT para reestruturação de transcrições, e o modelo SBERT para análise de similaridade semântica entre relatos, facilitando a identificação de conteúdos redundantes nos artigos coletados.

Destacam-se aqui os inúmeros métodos investigados e testados para cálculo de similaridade, que foi um ponto central deste trabalho de pesquisa. Inicialmente, a medida de Levenshtein foi utilizada para tratar todos os artigos coletados (textos e transcrições de vídeos). Contudo,

depois de vários testes com variações dessa medida, ficou claro que essa técnica não dava bons resultados para textos longos oriundos dos vídeos no YouTube. Assim, optou-se por usar a medida de Levenshtein apenas para tratar textos curtos.

A partir desse achado, buscamos outras alternativas, chegando até a solução adotada: ChatGPT para normalizar os textos longos e SBERT para realizar o cálculo de similaridade semântica entre esses artigos. Os resultados obtidos com essa combinação foram muito satisfatórios, tendo sido aprovados pelos colaboradores da Motorola usuários do sistema. Ainda foi investigado o uso do Gemini em substituição ao SBERT, porém os resultados foram muito semelhantes, e a escolha final foi o SBERT. O Gemini será novamente explorado em trabalhos futuros (Seção 6.3).

Os resultados demonstraram que a combinação dessas técnicas foi eficiente e viabilizou a extração automatizada de informações relevantes, superando abordagens manuais e ampliando o escopo de detecção de vulnerabilidades.

Questão de Pesquisa 2: *Como organizar, armazenar, atualizar e apresentar os dados extraídos aos usuários finais?*

Para tratar da organização, armazenamento, atualização e apresentação dos dados extraídos, foi desenvolvido um sistema composto por um banco de dados relacional (MySQL), *scripts* de automação de coleta e uma interface web interativa. Os dados foram estruturados e armazenados de forma organizada, com metadados extraídos dos textos coletados, permitindo atualizações contínuas e automáticas à medida que novos relatos são detectados.

A interface web desenvolvida possibilita uma visualização clara dos dados, com filtros, marcações colaborativas, mecanismos de busca, dashboards interativos e gráficos analíticos (como *heatmaps*, nuvens de palavras e gráficos de dispersão). Esses recursos facilitam a triagem, análise e priorização dos relatos técnicos por parte dos analistas de segurança, reduzindo o esforço manual e promovendo maior precisão no monitoramento de vulnerabilidades.

6.2 PRINCIPAIS CONTRIBUIÇÕES

O desenvolvimento e a validação do sistema protótipo apresentado nesta dissertação trouxeram avanços relevantes para a área de segurança em dispositivos Android, especialmente na detecção automatizada de vulnerabilidades relacionadas ao *Factory Reset Protection* (FRP). As principais contribuições deste trabalho incluem:

- **Arquitetura Modular e Extensível:** O sistema foi concebido com arquitetura desacoplada, organizada em módulos independentes (coleta, processamento, análise, armazenamento e visualização), o que facilita futuras expansões para outros tipos de vulnerabilidade, integração de novas fontes de dados ou substituição de técnicas analíticas.

- **Automação de Coleta e Pré-Processamento:** Foram desenvolvidos módulos específicos para extração automática de metadados, transcrições e conteúdos técnicos, com aplicação de técnicas robustas de web scraping e utilização de APIs públicas. O pré-processamento automatizado normalizou e enriqueceu os textos, viabilizando análises posteriores mais precisas.

- **Integração de Múltiplas Fontes de Dados:** O sistema propôs e implementou um pipeline capaz de coletar, processar e analisar informações provenientes de sites especializados, fóruns e vídeos do YouTube, superando limitações de abordagens restritas a uma única fonte e ampliando o escopo da detecção de vulnerabilidades.

- **Aplicação de PLN e Modelos de Linguagem:** O uso combinado de Processamento de Linguagem Natural, modelos de linguagem (ChatGPT), e análise de similaridade semântica (SBERT) permitiu padronizar, estruturar e comparar relatos provenientes de fontes heterogêneas, aumentando a precisão na identificação de conteúdos redundantes ou inéditos.

- **Inteface Web e Segurança:** A interface desenvolvida proporciona uma visualização clara, filtros avançados, mecanismos de marcação e comentários colaborativos, além de controle de acesso restrito. Esses recursos otimizam o trabalho dos analistas, tornando o processo de triagem mais ágil e seguro.

- **Validação e Análises Visuais:** O protótipo foi validado por meio de testes quantitativos e qualitativos, demonstrando sua efetividade na priorização de artigos relevantes e na redução do esforço manual. As técnicas visuais aplicadas (*heatmaps*, nuvens de palavras, gráficos de dispersão) facilitaram a compreensão dos resultados e dos agrupamentos temáticos encontrados.

Em síntese, este trabalho oferece uma solução inovadora, flexível e adaptável para apoiar equipes de segurança na detecção proativa de vulnerabilidades em dispositivos Android, ser-

vindo como referência tanto para aplicações práticas quanto para pesquisas futuras em mineração de texto e cibersegurança.

6.3 TRABALHOS FUTUROS

O trabalho de pesquisa desenvolvido no curso de mestrado abre espaço para múltiplas direções de continuidade e aprimoramento. Entre as principais possibilidades de trabalhos futuros, destacam-se:

- **Ampliação e Diversificação das Fontes de Dados:** Expandir a coleta para fóruns especializados, comunidades em redes sociais, repositórios de código, blogs técnicos e outras plataformas colaborativas. A automação de buscas em múltiplos domínios e canais do YouTube pode ampliar significativamente a cobertura e o potencial de descoberta de relatos relevantes.
- **Suporte Multilíngue:** Adaptar o pipeline para processar artigos e vídeos em diferentes idiomas ampliará o alcance internacional da ferramenta. O suporte multilíngue pode ser aprimorado tanto no pré-processamento quanto na escolha de modelos LLM e de similaridade, permitindo análises comparativas globais sobre vulnerabilidades.
- **Expansão para outros Fabricantes e Vulnerabilidades:** O sistema foi projetado de forma parametrizável, podendo tratar qualquer fabricante de dispositivos Android, e não apenas a Motorola. Dessa forma, pode ser facilmente ajustado para monitorar modelos de marcas como Samsung, Xiaomi, Huawei, entre outras. Além disso, o mesmo mecanismo pode ser expandido para novas categorias de vulnerabilidades no ecossistema Android, como exploits de firmware, permissões indevidas ou falhas em aplicativos, ampliando o escopo de detecção e análise sem necessidade de alterações estruturais no código.
- **Seleção e Avaliação de outros Modelos LLM:** Pesquisas futuras podem explorar diferentes modelos de linguagem (LLMs) na reestruturação de transcrições, bem como estratégias de uso combinado (ensemble) para aumentar a robustez e a acurácia dos resultados.
- **Automação e Escalabilidade:** A inclusão de rotinas automáticas de coleta, processamento e notificações em tempo real pode transformar o sistema em uma plataforma de monitoramento contínuo, reduzindo a necessidade de intervenção manual.

- **Aprimoramento da Interface e Visualização dos Dados:** O desenvolvimento de dashboards interativos, relatórios customizados e integrações com ferramentas corporativas pode tornar o sistema mais acessível e útil para equipes técnicas e gestores.
- **Publicação, Compartilhamento e Colaboração:** A disponibilização de *datasets* anonimizados e do código-fonte pode impulsionar colaborações acadêmicas, *benchmarking* e inovação aberta em mineração de texto e segurança digital.

Essas direções revelam o grande potencial deste trabalho para contribuir de forma contínua com a inovação em segurança de dispositivos móveis, acompanhando a evolução das ameaças e das tecnologias do setor. Ressalta-se, ainda, que, conforme destacado na introdução, a arquitetura modular do sistema permite sua adaptação para diferentes famílias de dispositivos e sistemas operacionais, bastando ajustes nos parâmetros de entrada. Essa flexibilidade amplia significativamente o potencial de aplicação da solução desenvolvida, tornando-a apta a atender novos cenários e desafios emergentes na área de segurança da informação.

Por fim, este trabalho consolida-se como uma relevante contribuição acadêmica e tecnológica, desenvolvida no âmbito do Centro de Informática da UFPE no contexto da parceria estratégica com o projeto CIn-Motorola e sua equipe de segurança. Ao propor, implementar e validar uma solução inovadora para identificação automatizada de vulnerabilidades em dispositivos Android, reafirma-se o compromisso institucional com a excelência em pesquisa aplicada e com a transferência de conhecimento para o setor produtivo.

Espera-se que as metodologias, ferramentas e reflexões aqui apresentadas sirvam de referência para novas iniciativas acadêmicas, fomentem a colaboração entre universidade e indústria e inspirem o contínuo aprimoramento das práticas relacionadas à proteção de dados e sistemas, sempre pautadas por rigor técnico, responsabilidade ética e contribuição social.

REFERÊNCIAS

AGGARWAL, C. C.; ZHAI, C. A survey of text clustering algorithms. In: AGGARWAL, C. C.; ZHAI, C. (Ed.). *Mining Text Data*. [S.l.]: Springer, 2012. p. 77–128.

ARANHA, C. N. *Uma abordagem de pré-processamento automático para mineração de textos em português: Sob o enfoque da inteligência computacional*. Tese (Doutorado) — Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio), Rio de Janeiro, 2007.

ATHIWARATKUN, B.; WILSON, A. G.; ANANDKUMAR, A. Probabilistic fasttext for multi-sense word embeddings. *arXiv preprint*, 2018.

AZIZ, A. *Longest Common Subsequence*. 1965. Dynamic programming. Disponível em: <https://users.ece.utexas.edu/~adnan/360C/dp.pdf>.

BAEZA-YATES, R.; RIBEIRO-NETO, B. *Modern Information Retrieval: The Concepts and Technology Behind Search*. 2nd. ed. [S.l.]: Addison-Wesley, 2013.

BAYAZIT, E. C.; SAHINGOZ, O. K.; DOGAN, B. A deep learning based android malware detection system with static analysis. In: IEEE. *2022 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)*. [S.l.], 2022. p. 1–6.

CHEN, T.; MAO, Q.; YANG, Y.; LV, M.; ZHU, J. Tinydroid: a lightweight and efficient model for android malware detection and classification. *Mobile information systems*, Wiley Online Library, v. 2018, n. 1, p. 4157156, 2018.

DEVLIN, J.; CHANG, M.-W.; LEE, K.; TOUTANOVA, K. BERT: Pre-training of deep bidirectional transformers for language understanding. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. [s.n.], 2019. p. 4171–4186. Disponível em: <https://aclanthology.org/N19-1423/>.

ENCK, W.; GILBERT, P.; HAN, S.; TENDULKAR, V.; CHUN, B.-G.; COX, L. P.; JUNG, J.; MCDANIEL, P.; SHETH, A. N. Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. *ACM Transactions on Computer Systems (TOCS)*, ACM New York, NY, USA, v. 32, n. 2, p. 1–29, 2014. Disponível em: <https://dl.acm.org/doi/abs/10.1145/2619091>.

ENCK, W.; OCTEAU, D.; MCDANIEL, P. D.; CHAUDHURI, S. A study of android application security. In: *USENIX security symposium*. [S.l.: s.n.], 2011. v. 2, n. 2, p. 1–38.

FARUKI, P.; BHARMAL, A.; LAXMI, V.; GANMOOR, V.; GAUR, M. S.; CONTI, M.; RAJARAJAN, M. Android security: a survey of issues, malware penetration, and defenses. *IEEE communications surveys & tutorials*, IEEE, v. 17, n. 2, p. 998–1022, 2015.

GEEKSFORGEEKS. *Software Development Life Cycle (SDLC)*. 2025. Acesso em: [02 de junho de 2025]. Disponível em: <https://www.geeksforgeeks.org/software-development-life-cycle-sdlc/>.

GHORBANI, N.; JABBARVAND, R.; SALEHNAMADI, N.; GARCIA, J.; MALEK, S. Deltadroid: Dynamic delivery testing in android. *ACM Trans. Softw. Eng. Methodol.*,

Association for Computing Machinery, v. 32, n. 4, maio 2023. ISSN 1049-331X. Disponível em: <https://doi.org/10.1145/3563213>.

GRISHMAN, R. Information extraction. *IEEE Intelligent Systems*, v. 30, n. 5, p. 8–15, 2015.

HOODA, I.; CHHILLAR, R. S. Software test process, testing types and techniques. *International Journal of Computer Applications*, Foundation of Computer Science, v. 111, n. 13, 2015.

IBM. *Text Mining*. 2025. IBM Corporation. Disponível em: <https://www.ibm.com/br-pt/think/topics/text-mining>.

IBM. *Text Mining Use Cases*. 2025. IBM Corporation. Disponível em: <https://www.ibm.com/br-pt/think/topics/text-mining-use-cases>.

IBM. *Word Embeddings*. 2025. IBM Corporation. Disponível em: <https://www.ibm.com/think/topics/word-embeddings>.

JOHNSON, S. J.; MURTY, M. R.; NAVAKANTH, I. A detailed review on word embedding techniques with emphasis on word2vec. *Multimedia Tools and Applications*, v. 83, n. 13, p. 37979–38007, 2024.

JURAFSKY, D.; MARTIN, J. H. *Speech and Language Processing (2nd Edition)*. USA: Prentice-Hall, Inc., 2009. ISBN 0131873210. Disponível em: <https://home.cs.colorado.edu/~martin/slp.html>.

JURAFSKY, D.; MARTIN, J. H. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models*. 3rd. ed. [s.n.], 2025. Online manuscript released January 12, 2025. Disponível em: <https://web.stanford.edu/~jurafsky/slp3/>.

KAMP, S.; FAYAZI, M.; BENAMEUR-EL, Z.; YU, S.; DRESLINSKI, R. Open information extraction: A review of baseline techniques, approaches, and applications. *arXiv preprint*, 2023.

KARIM, A.; CHANG, V.; FIRDAUS, A. Android botnets: a proof-of-concept using hybrid analysis approach. *Journal of Organizational and End User Computing (JOEUC)*, IGI Global, v. 32, n. 3, p. 50–67, 2020.

KELLEY, K. *Vulnerability in Security: A Complete Overview*. 2023. Simpli Learn. Disponível em: <https://www.simplilearn.com.cach3.com/vulnerability-in-security-article.html>.

KOWSARI, K.; MEIMANDI, K. J.; HEIDARYSAFA, M.; MENDU, S.; BARNES, L.; BROWN, D. Text classification algorithms: A survey. *Information*, v. 10, n. 4, p. 150, 2019.

LANG, M. Analysis of android factory resets. Viena, Austria, 2022. Disponível em: <https://appsec.at/publication/lang-2022-analysis/lang-2022-analysis.pdf>.

LEE, S.; JUNG, Y.; LEE, J.; LEE, B.; KWON, T. T. Android remote unlocking service using synthetic password: A hardware security-preserving approach. In: IEEE. *2021 IEEE Secure Development Conference (SecDev)*. 2021. p. 63–70. Disponível em: <https://ieeexplore.ieee.org/abstract/document/9652635>.

LEVENSHTEIN, V. I. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, v. 10, n. 8, p. 707–710, 1966. Disponível em: <https://nymity.ch/sybilhunting/pdf/Levenshtein1966a.pdf>.

LI, Y.; MCLEAN, D.; BANDAR, Z. A.; O'SHEA, J. D.; CROCKETT, K. Sentence similarity based on semantic nets and corpus statistics. *IEEE Transactions on Knowledge and Data Engineering*, v. 18, n. 8, p. 1138–1150, 2006.

MIKOLOV, T.; CHEN, K.; CORRADO, G.; DEAN, J. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, v. 3781, 2013.

MYERS, G. J.; SANDLER, C.; BADGETT, T. *The art of software testing*. [S.l.]: John Wiley & Sons, 2011.

PRAKOSO, D. W.; ABDI, A.; AMRIT, C. Short text similarity measurement methods: A review. *Soft Computing*, v. 25, p. 4699–4723, 2021.

SCHAIK, T. A. van; PUGH, B. A field guide to automatic evaluation of LLM-generated summaries. In: *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*. [S.l.: s.n.], 2024. p. 2832–2836.

SENANAYAKE, J.; KALUTARAGE, H.; AL-KADRI, M. O.; PETROVSKI, A.; PIRAS, L. Android source code vulnerability detection: A systematic literature review. *ACM Comput. Surv.*, Association for Computing Machinery, v. 55, n. 9, 2023.

SHABTAI, A.; KANONOV, U.; ELOVICI, Y.; GLEZER, C.; WEISS, Y. "andromaly": a behavioral malware detection framework for android devices. *Journal of Intelligent Information Systems*, Springer, v. 38, n. 1, p. 161–190, 2012.

SOMMERVILLE, I. *Software Engineering*. 10th. ed. [S.l.]: Pearson Education Inc, 2015.

WANKHADE, M.; RAO, A. C. S.; KULKARNI, C. A survey on sentiment analysis methods, applications, and challenges. *Artificial Intelligence Review*, v. 55, n. 7, p. 5731–5780, 2022.

XU, D.; CHEN, W.; PENG, W.; ZHANG, C.; XU, T.; ZHAO, X.; CHEN, E. Large language models for generative information extraction: A survey. *Frontiers of Computer Science*, v. 18, n. 6, p. 186–357, 2024.

ZHAO, B. Web scraping. In: SCHINTLER, L. A.; MCNEELY, C. L. (Ed.). *Encyclopedia of Big Data*. [S.l.]: Springer, 2017.

ZHOU, Y.; JIANG, X. Dissecting android malware: Characterization and evolution. In: IEEE. *2012 IEEE symposium on security and privacy*. [S.l.], 2012. p. 95–109.