UNIVERSIDADE FEDERAL DO PAMPA
CAMPUS BAGÉ
CURSO DE ENGENHARIA DE COMPUTAÇÃO

LUCAS VILANOVA BARCELLOS

ADBUILDER - UMA FERRAMENTA DE CONSTRUÇÃO DE *DATASETS* PARA DETECÇÃO DE *MALWARES* ANDROID

Bagé
2023

LUCAS VILANOVA BARCELLOS

ADBUILDER - UMA FERRAMENTA DE CONSTRUÇÃO DE *DATASETS* PARA DETECÇÃO DE *MALWARES* ANDROID

Trabalho de Conclusão de Curso apresentado ao curso de Bacharelado em Engenharia de Computação como requisito parcial para a obtenção do grau de Bacharel em Engenharia de Computação.

Orientador: Érico Marcelo Hoff do Amaral
Coorientador: Diego Luis Kreutz

Bagé
2023

Ficha catalográfica elaborada automaticamente com os dados fornecidos
pelo(a) autor(a) através do Módulo de Biblioteca do
Sistema GURI (Gestão Unificada de Recursos Institucionais) .

V696a Vilanova, Lucas Barcellos
ADBuilder: uma ferramenta de construção de datasets para detecção de malwares Android / Lucas Barcellos Vilanova.
116 p.

Trabalho de Conclusão de Curso(Graduação)-- Universidade Federal do Pampa, ENGENHARIA DE COMPUTAÇÃO, 2023.
"Orientação: Érico Marcelo Hoff do Amaral".

1. Construção de datasets. 2. Detecção de malwares. 3. Ferramenta. 4. Android. 5. Características. I. Título.

LUCAS VILANOVA BARCELLOS

ADBUILDER - UMA FERRAMENTA DE CONSTRUÇÃO DE DATASETS PARA DETECÇÃO DE MALWARES ANDROID

Trabalho de Conclusão de Curso apresentado ao curso de Bacharelado em Engenharia de Computação como requisito parcial para a obtenção do grau de Bacharel em Engenharia de Computação.

Dissertação defendida e aprovada em: 14 de julho de 2023.

Banca examinadora:

Prof. Dr. Érico Marcelo Hoff do Amaral
Orientador
UNIPAMPA

Prof. Dr. Ewerton Rodrigues Andrade
UNIR

Prof. Dr. Gerson Alberto Leiria Nunes
UNIPAMPA

Assinado eletronicamente por **ERICO MARCELO HOFF DO AMARAL**, **PROFESSOR DO MAGISTERIO SUPERIOR**, em 19/07/2023, às 21:38, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

Assinado eletronicamente por **Ewerton Rodrigues Andrade**, **Usuário Externo**, em 20/07/2023, às 15:49, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

A autenticidade deste documento pode ser conferida no site https://sei.unipampa.edu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **1187888** e o código CRC **40523621**.

Dedico este trabalho à minha família, que sempre esteve presente e me apoiou em todas as etapas desta jornada. Aos meus pais, que acreditaram em mim e me incentivaram a seguir meus sonhos, e a minha irmã, que foi minha confidente e amiga. Dedico também ao meu orientador, co-orientador e aos meus colegas de projeto, que contribuíram com suas experiências e conhecimentos para o desenvolvimento da ferramenta ADBuilder.

AGRADECIMENTO

Primeiramente, agradeço a Deus pelo dom da vida, saúde e inteligência que me permitiram levar adiante este trabalho. Agradeço também por todas as bênçãos e oportunidades que Ele me proporcionou ao longo desta jornada. Peço a Sua contínua orientação e bençãos para minha vida e carreira. Agradeço ao meu orientador e co-orientador, pelo apoio, dedicação e paciência durante todo o desenvolvimento deste trabalho. Agradeço também a instituição de ensino, que me proporcionou as condições necessárias para a realização deste estudo, e a todos os professores e amigos que contribuíram para minha formação. Por fim, agradeço aos colegas de projeto, que contribuíram com sugestões e insights valiosos durante a construção da ferramenta ADBuilder.

“If I have seen farther than others, it is because I stood on the shoulders of giants.”
— Sir Isaac Newton

RESUMO

A importância de *datasets* precisos e eficazes para ferramentas que realizam análises de *malwares* e de sistemas operacionais Android são fundamentais. Através de uma investigação, identificou-se a necessidade de soluções mais eficientes nesta área. Com este propósito, foi proposta a construção automatizada de um *dataset*, que integra processos como o *download* automatizado de APKs, a extração de características, a rotulação dos aplicativos, a limpeza de dados e a geração final do *dataset*, formando assim a ferramenta ADBuilder.

A arquitetura da ferramenta é baseada em 4 (quatro) módulos independentes e fracamente acoplados, seguindo o princípio de produtor e consumidor, que se comunicam através de filas. Através de estudos iniciais, verificou-se a viabilidade de implementação da ferramenta e foram desenvolvidos os dois primeiros módulos, o *download* e a extração.

O módulo de *download* foi implementado utilizando o repositório AndroZoo, para obter aplicativos Android de forma rápida e eficiente. Já o módulo de extração foi realizado com a utilização da ferramenta Androguard, que é capaz de extrair as características de um aplicativo Android. Os resultados obtidos mostraram que a integração dos dois primeiros módulos apresentou resultados satisfatórios, com a funcionalidade esperada de baixar um aplicativo e extrair suas características.

Dessa forma, os resultados dessa pesquisa foram promissores e validaram a implementação dos módulos subsequentes da ferramenta ADBuilder, que são o módulo de rotulação e o módulo de geração. A implementação desses módulos demonstrou eficiência ao obter os dados de rotulação dos aplicativos analisados, através do serviço do VirusTotal, e criar o *dataset* final. Através desses resultados, foi comprovado que a ADBuilder é capaz de integrar com sucesso os processos de rotulação dos aplicativos, a limpeza dos dados e a geração final do *dataset*. A comparação com o conjunto de dados Drebin-215 evidenciou a amplitude e a quantidade de características extraídas pela ADBuilder, reforçando sua capacidade de descrever de forma abrangente o comportamento de aplicações maliciosas. Esses resultados validam a importância da ADBuilder como uma ferramenta eficiente para a análise e detecção de *malwares* Android, impulsionando futuros avanços e aprimoramentos nessa área.

Palavras-chave: *Datasets*. Android. Ferramenta. Construção. Extração. Rotulação. Características. Detecção. *Malwares*.

ABSTRACT

The importance of accurate and effective datasets for tools that perform analysis of malware and Android operating system is crucial. Through an investigation, the need for more efficient solutions in this area was identified. With this purpose in mind, the automated construction of datasets was proposed, which integrates processes such as automated APK downloading, feature extraction, app labeling, data cleaning, and final dataset generation, forming the ADBuilder tool.

The tool architecture is based on 4 (four) independent and weakly coupled modules, following the producer-consumer principle, that communicate through queues. Through initial studies, the feasibility of the tool implementation was verified and the first two modules, downloading and extraction, were developed.

The download module was implemented using the AndroZoo repository, to obtain Android apps quickly and efficiently. The extraction module was carried out using the Androguard tool, which is capable of extracting the features of an Android app. The results obtained showed that the integration of the first two modules had satisfactory results, with the expected functionality of downloading an app and extracting its features. Therefore, the results of this research were promising and validated the implementation of the subsequent modules of the ADBuilder tool, namely the labeling module and the generation module. The implementation of these modules demonstrated efficiency in obtaining the labeling data of the analyzed applications, through the VirusTotal service, and creating the final dataset. Through these results, it was proven that ADBuilder is capable of successfully integrating the application labeling processes, data cleaning, and final dataset generation. The comparison with Drebin-215 dataset highlighted the breadth and quantity of features extracted by ADBuilder, reinforcing its ability to comprehensively describe the behavior of malicious applications. These results validate the importance of ADBuilder as an efficient tool for the analysis and detection of Android malware, driving future advancements and enhancements in this field.

Keywords: Datasets. Android. Tool. Construction. Extraction. Labelling. Features. Detection. Malwares.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ABNT	Associação Brasileira de Normas Técnicas
ANPD	Autoridade Nacional de Proteção de Dados
API	Application Programming Interface (Interface de Programação de Aplicação)
APK	Android Application Package Kit
ART	Android Runtime
CG	Call Graph
CSV	Comma Separated Values (Arquivo com Valores Separados por Vírgula)
DEX	Dalvik Executable
FIFO	First in, First Out
GPP	Google Play Protect
HAL	Hardware Abstraction Layer
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
IU	Interface de Usuário
LGPD	Lei Geral de Proteção de Dados Pessoais
UNIPAMPA	Universidade Federal do Pampa

SUMÁRIO

1 INTRODUÇÃO

Os *datasets* são a base de inteligência de um algoritmo de aprendizado de máquina (JAIN, 2020). Dessa forma, é preciso fornecer *datasets* de qualidade para treinar e validar modelos preditivos eficazes para detectar *malwares* Android. Vale ressaltar que tanto a quantidade de amostras e características quanto a qualidade e atualidade dos dados impactam fortemente o desempenho dos modelos criados, conforme abordado nos trabalhos (ALLIX, 2015), (WANG et al., 2019a) e (VILANOVA et al., 2021). O problema é que a grande maioria dos *datasets* existentes falham em atenderem a esses requisitos, os quais abrangem a quantidade e qualidade das amostras, atualidade dos dados e a correta rotulação das amostras (i.e., entre benigna ou maliciosa). Além disso, muitos *datasets* disponíveis publicamente contém vieses nos dados (i.e., dados com ruídos que não foram bem tratados e que podem confundir o discernimento do modelo preditivo ao tomar uma decisão), abrangendo amostras duplicadas, valores inválidos (e.g., valores faltantes), dados defasados (i.e., dados de aplicativos antigos) e defasagem na rotulagem, carecendo de uma etapa de limpeza dos dados.

Um exemplo é o *dataset* Drebin (ARP et al., 2014), amplamente utilizado na literatura, que além de conter dados defasados e amostras duplicadas, contém erros de rotulação das amostras (e.g., um aplicativo malicioso rotulado como benigno e vice-versa), devido a falta de atualização dos dados. Outro estudo aponta sobre erros de marcação de amostras nos *datasets* MalGenome, Drebin-215, Piggybacking e AMD (WANG et al., 2019a) e sobre o problema na defasagem na rotulagem dos aplicativos (LASHKARI et al., 2018).

Em um levantamento bibliográfico realizado[1], foi constatado que a maioria dos trabalhos voltados a construção de *datasets* para detecção de *malwares* Android carecem de: etapas cruciais como a consolidação de características e limpeza de dados do *dataset*; informações detalhadas da extração de características e rotulação das amostras; o limiar para rotular um *malware*; e um sistema automatizado e integrado capaz de construir conjuntos de dados atualizados (VILANOVA et al., 2022). Alguns exemplos são os trabalhos (LASHKARI et al., 2018; WANG et al., 2019a; MAHINDRU et al., 2020; DÜZGÜN et al., 2021), que além de não disponibilizarem o código e nem detalhar o ferramental utilizado, não incorporam etapas de consolidação de características e limpeza de dados do *dataset*. Essas limitações podem fazer com que o conjunto de

[1] https://docs.google.com/spreadsheets/d/11qyXrkwIY3GuTfPw6NaXV6MnqIySQDqS/edit?usp=sharing&ouid=109051153975920523864&rtpof=true&sd=true

dados contenha dados duplicados, valores faltantes ou inválidos, número reduzido ou excessivo de características e erros de nomenclatura (e.g., mesma característica com nomes diferentes) (VILANOVA et al., 2022).

A necessidade de se construir um conjunto de dados de qualidade que seja adequado para treinar modelos preditivos é indispensável, para isso é preciso executar algumas etapas principais, como extrair as características de um aplicativo Android e rotulá-lo (i.e., entre benigno ou maligno). Porém, como já discutido, os *datasets* disponíveis publicamente carecem dessas etapas. Por exemplo, na etapa da extração de características, observam-se algumas limitações. Dentre os trabalhos correlatos levantados, voltados para análise estática, observou-se que nenhum utilizou a ferramenta Androguard (KUMAR; YADAV; SINGH, 2022), que pode ser considerada uma das ferramentas mais completas e eficazes para a extração de características estáticas, de acordo com o levantamento realizado por (PONTES et al., 2021). Até mesmo os trabalhos que utilizam apenas características dinâmicas, como (LASHKARI et al., 2018) e (CATAK; YAZI, 2019), falham em discutir e analisar qualitativamente as diferentes ferramentas disponíveis para extrair as características dos aplicativos. De acordo com os trabalhos (PAN et al., 2020) e (PONTES et al., 2021), é fundamental realizar uma avaliação qualitativa sobre as ferramentas de extração de características disponíveis e utilizar as mais adequadas para cada caso de uso.

Diante da carência de *datasets* públicos de qualidade, este trabalho propõe criar uma ferramenta capaz de construir conjuntos de dados significativos para serem utilizados na detecção de *malwares* para o sistema Android, no qual foi batizada como ADBuilder. A ADBuilder é uma ferramenta integrada e automatizada capaz de construir conjuntos de dados atualizados, que possuam uma diversidade de características relevantes para treinar os modelos preditivos de aprendizado de máquina, incorporando etapas de validação e limpeza de dados do *dataset*.

1.1 Problema de Pesquisa

No decorrer dos anos, as estratégias de ataques à dispositivos Android foram sofrendo evoluções, em consequência, as aplicações maliciosas tornaram-se mais sofisticadas, alterando seu comportamento e nível de complexidade, resultando em uma dificuldade maior em detectá-las. Os trabalhos (TAM, 2017) e (SAHAY, 2020), abordam como essas aplicações evoluem e impactam no aprendizado de máquina. Sabendo disso,

os algoritmos de aprendizado de máquina não conseguem acompanhar essa evolução devido a escassez de *datasets* de qualidade, que sejam capazes de fornecer dados representativos e atuais aos modelos preditivos. Neste contexto o presente trabalho tem como problema de pesquisa: "É possível implementar uma ferramenta integrada e automatizada capaz de construir conjuntos de dados atualizados para detecção de *malwares* no Android, com o intuito de disponibilizá-los publicamente em um repositório Github de livre acesso?"

1.2 Objetivo Geral

Implementar uma ferramenta integrada e automatizada capaz de construir conjuntos de dados atualizados para detecção de *malwares* para o sistema operacional Android, disponibilizando todos os dados e ferramental desenvolvidos durante o processo.

1.3 Objetivos Específicos

Este trabalho possui como objetivos específicos:

1. Definir o problema de pesquisa;
2. Realizar o levantamento do referencial teórico e dos trabalhos correlatos do tema;
3. Pesquisar sobre elementos que poderão ser integradas na ADBuilder;
4. Construir a arquitetura e modelagem da ferramenta;
5. Implementar, testar e validar a ferramenta;
6. Analisar e descrever os resultados obtidos;
7. Escrita da documentação e manuais da ferramenta;
8. Escrita do TCC II;
9. Entrega da versão para a banca do TCC II;
10. Entrega da versão final para a banca do TCC II.

1.4 Organização do trabalho

No Capítulo 1 são apresentados a introdução da proposta da ferramenta ADBuilder, o problema de pesquisa, o objetivo geral e os objetivos específicos. No Capítulo 2 é apresentado uma descrição detalhada da abordagem utilizada para conduzir o estudo e alcançar os objetivos propostos. No Capítulo 3 é apresentado uma revisão da literatura existente sobre o tema estudado, no qual baseou-se em levantar os conhecimentos sobre computação móvel, segurança da informação e de sistemas, sistema operacional Android, *datasets* e aprendizado de máquina. Em seguida, são apresentados os trabalhos correlatos a esta pesquisa. No Capítulo 4 é apresentado a proposta da ferramenta ADBuilder, no qual são abordados: a descrição da proposta; a validação da solução; a análise e modelagem da ferramenta, no qual são levantados os requisitos funcionais, não funcionais e de sistema, casos de uso, diagrama de classes conceitual, diagrama de sequência, e como os dados serão armazenados; as tecnologias que serão empregadas, como o repositórios de APKs, serviços de rotulação de aplicativos, e ferramentas de extração de características; a implementação da ADBuilder; os resultados e discussões; e a validação da ferramenta. No Capítulo 5 são discutidas as considerações finais do trabalho. Por fim, no Capítulo 6 são apresentados os trabalhos futuros.

2 METODOLOGIA

A classificação da pesquisa é importante para apresentar o direcionamento que o trabalho tomou. A Figura 1 ilustra a classificação que representa a pesquisa.

Figura 1 – Classificação da pesquisa.

Fonte: Autor (2022).

Segundo (SILVEIRA; CÓRDOVA, 2009), a pesquisa foi classificada como aplicada porque teve como finalidade gerar conhecimentos para aplicações práticas com o objetivo de solucionar problemas específicos. Sendo assim, a implementação da ferramenta foi composta por conhecimentos que foram necessários para gerar o produto final (e.g., conhecimentos sobre características dos aplicativos Android), juntando teoria e prática para solucionar um problema específico da área de detecção de *malwares* Android.

A abordagem do problema que classifica a pesquisa foi mista (i.e., quanti-qualitativa). A abordagem qualitativa tem como foco a interpretação e compreensão dos fenômenos ou objeto. Sendo assim, essa abordagem foi qualificada para essa pesquisa, pois foi muito importante para entender cada processo (módulo) da ferramenta (e.g., como era realizada a extração de características e rotulação das amostras), assim como analisar a qualidade da ferramenta e dos *datasets* construídos. Em relação da abordagem ser quantitativa, além de traduzir em números os dados coletados, foram utilizadas tabelas para apresentar as estatísticas da ferramenta (e.g., tempo de execução e consumo de memória), com a finalidade de interpretar o funcionamento e qualidade da mesma, tornando a abordagem do problema como quanti-qualitativa.

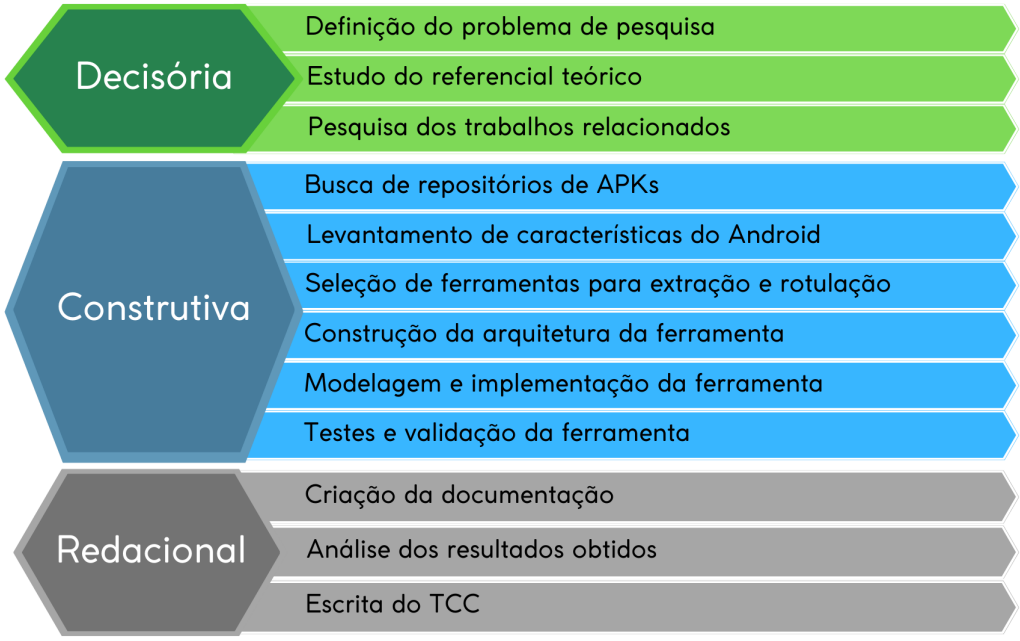
Em relação aos objetivos, a pesquisa foi classificada como exploratória. A pesquisa exploratória busca explorar os conceitos e ideias de uma determinada área, a fim de definir as informações e escolhas que melhor se adéquam ao projeto. Nesse contexto,

a pesquisa exploratória se enquadra neste trabalho, pois para realizar a implementação da ferramenta foram necessários explorar e levantar informações bibliográficas para analisar e selecionar os elementos com mais aderência para serem integrados a este projeto. Foi preciso buscar informações, como quais características do Android existem e poderiam ser utilizadas no aprendizado de máquina, quais tecnologias (e.g., ferramenta de extração de características) e etapas (e.g., limpeza dos dados, filtragem de características) seriam requisitos para construir os conjuntos de dados, por exemplo. Além disso, foi necessário buscar conhecimentos básicos sobre engenharia de *software* para seguir um planejamento adequado para a construção da ferramenta.

Em relação aos procedimentos técnicos, a pesquisa foi classificada como bibliográfica e experimental. Antes de partir para a implementação, foi necessário realizar um levantamento do referencial teórico do tema, além de buscar trabalhos relacionados e informações que auxiliem no processo de entendimento do problema e construção da solução. Após essa etapa é que entrou a pesquisa experimental, que foi: propor a solução do problema, a ferramenta que visa construir conjuntos de dados de forma automatizada; implementar a proposta; realizar experimentos; e por fim, validar se a solução era viável.

O planejamento desta pesquisa dependeu basicamente de três fases: (a) decisória - referente à escolha do tema, à definição e à delimitação do problema de pesquisa; (b) construtiva - referente à construção de um plano de pesquisa e à execução da pesquisa propriamente dita; e (c) redacional - referente à análise dos dados e informações obtidas na fase construtiva (SILVA; MENEZES, 2005). Então, para cumprir com os objetivos dessa pesquisa, foi necessário com que a organização do trabalho fosse planejada como uma metodologia que siga as etapas apresentadas na Figura 2.

Figura 2 – Etapas do desenvolvimento da pesquisa.



Fonte: Autor (2022).

Observando a Figura 2, a metodologia da pesquisa iniciou-se pela fase decisória que consistiu em definir o problema de pesquisa. Nessa etapa, foi definido o problema que o trabalho se propôs a solucionar. A decisão do problema foi baseada pela falta de *datasets* qualitativos atualizados disponíveis para detecção de *malwares* no Android, buscando contribuir com a sociedade científica, dado que qualquer pesquisador ou interessado poderá construir seu próprio conjunto de dados, a partir dos aplicativos Android de sua escolha. Depois dessa etapa, fez-se necessário realizar uma pesquisa bibliográfica, levantando informações do referencial teórico no qual esse trabalho foi baseado, para ajudar no entendimento do problema e dos requisitos para a construção da solução. Por fim, ao final dessa fase, foram analisados os trabalhos relacionados que se mostram relevantes para a pesquisa.

No início da fase construtiva foi necessário realizar o levantamento de informações específicas na área de domínio do problema. Primeiramente, foi importante levantar informações sobre repositórios de aplicativos Android, que permitem o *download* dos APKs de forma automatizada. Em seguida, um estudo sobre as características dos aplicativos Android foi realizado, através de artigos e *datasets*. Após o estudo das características estar consolidado, foi realizado o levantamento de ferramentas que realizam a extração de características estáticas dos aplicativos Android, buscando selecionar a com melhor aderência ao projeto. Segundo (PONTES et al., 2021), as

características estáticas do Android Package (APK) são extraídas sem necessidade de execução do aplicativo. A abordagem de análise estática é a mais utilizada para detecção de aplicativos maliciosos, pois consome tempo e recursos computacionais menores quando em comparação à outras técnicas (WANG et al., 2019b) (e.g., a análise dinâmica, que requer que o aplicativo esteja em execução). Ao final da fase construtiva, foi realizado o levantamento de ferramentas ou serviços que forneçam dados de rotulação dos aplicativos Android. A busca teve como objetivo encontrar a melhor solução, que disponibilize o ferramental ou API que permita a utilização do serviço.

Após o levantamento de informações, iniciou-se a parte prática da fase construtiva, onde de fato o primeiro passo foi dado para a construção da ferramenta proposta. Primeiramente, foi necessário construir a arquitetura da ferramenta, tendo em vista que ela foi composta por 4 (quatro) módulos integrados (i.e., *download*, extração, rotulação e geração). Depois dessa etapa, foi necessário realizar a modelagem da ferramenta. Após isso, a implementação da ferramenta foi realizada utilizando as tecnologias com melhor aderência. Ao final dessa fase, foram realizados testes para validar se a solução da ferramenta era viável.

A última fase da metodologia é a redacional, a qual foi iniciada com a escrita da documentação da ferramenta e levantamento das dependências necessárias para sua execução. Em seguida, foram analisados todos os resultados obtidos das etapas anteriores, para então iniciar a escrita do Trabalho de Conclusão de Curso.

3 REFERENCIAL TEÓRICO

Neste capítulo será apresentado o fundamento teórico necessário para o entendimento e desenvolvimento do projeto. Primeiramente, a seção 3.1 abordará sobre a computação móvel em geral, nos dias de hoje. A seção 3.2, serão apresentados conceitos sobre segurança e sua importância em dispositivos móveis. A seção 3.3, será abordado sobre o sistema operacional Android. A seção 3.4, a discussão será sobre o que são os *datasets* e o aprendizado de máquina. Por fim, a seção 3.5 apresentará uma discussão sobre os trabalhos correlatos a esta pesquisa.

3.1 Computação móvel

Desde os anos 90 até os dias de hoje, nota-se que o desenvolvimento da tecnologia cresceu exponencialmente. A tecnologia foi muito investida e aplicada na área de computação móvel, no qual abrange o desenvolvimento para comunicação de celulares móveis, comunicação via satélite e redes locais sem fio. Essas tecnologias evoluíram e tornaram-se cada vez mais populares, permitindo que o acesso à informações a partir de qualquer lugar e a qualquer hora fosse possível para as pessoas (FIGUEIREDO; NAKAMURA, 2003).

A computação móvel permite aos usuários a terem acesso a serviços e aplicações, independentemente de suas localizações ou se os mesmos estão em movimento. Ademais, com o grande avanço da tecnologia, o desenvolvimento de novos dispositivos móveis, como *smartphones*, *tablets* e *notebooks* tornaram-se cada vez mais necessários no cotidiano das pessoas, o que levou a restrições no consumo de energia por parte do computador móvel, que conduziu ao desenvolvimento de trabalhos em diferentes áreas, como o gerenciamento de dados e sistemas operacionais (ITO; FERREIRA; SANT'ANA, 2003). Por exemplo, hoje, os *smartphones* possuem inúmeras aplicações que fornecem funcionalidades e serviços aos usuários, tendo em vista que todos dispositivos possuem um sistema operacional que os gerenciam (e.g., Android e IOS).

Segundo (MAZIERO, 2014), um sistema operacional é composto por diversas camadas de *software* que se integram para garantir o funcionamento do sistema de computação. A camada mais fundamental é o núcleo *(kernel)*, que é responsável pelo gerenciamento de recursos do sistema, como processos, memória, entrada e saída de dados, entre outros. Além disso, existe a camada de modo de usuário, que é responsável

por fornecer interfaces para que os usuários e aplicativos possam interagir com o sistema. Outras camadas incluem o gerenciador de arquivos, o gerenciador de dispositivos, o gerenciador de rede e o gerenciador de segurança. Essas camadas funcionam juntas para garantir que o sistema operacional seja capaz de executar várias tarefas simultaneamente e fornecer uma interface amigável para os usuários.

Em relação ao objetivo dos sistemas operacionais, "o problema de minimizar o consumo de energia é fundamental. Uma das questões mais importantes é como o estado do sistema deve ser salvo periodicamente para prevenir uma perda do estado no caso de falta de energia" (MATEUS; LOUREIRO, 1998). Além disso, um sistema operacional móvel é utilizado em equipamentos portáteis, como *smartphones* e *tablets*. Portanto, é necessário que haja: (a) uma gestão de energia que seja eficiente para utilizar a bateria do dispositivo; (b) a capacidade de conectividade em diversos tipos de rede (e.g., wifi e Bluetooth); e (c) a interação com uma variedade de sensores (e.g., GPS, tela de toque e leitor de digitais).

Além da computação móvel, o avanço do armazenamento na nuvem, a redução do custo de *hardware* e a expansão das redes WAN e LAN sem fio, permitiram que diversas áreas de negócios e serviços utilizassem essas tecnologias e os dispositivos móveis para facilitar o acesso à informação e manter os dados seguros. Por exemplo, se acontecer algo com o dispositivo do usuário (i.e., *hardware*), ele tem a segurança de que seus arquivos e dados importantes estão salvos na nuvem, mantendo sempre um *backup* atualizado dos mesmos. Além disso, a área de computação abrange todas demais áreas profissionais, pois ela visa facilitar e auxiliar o progresso tecnológico das mesmas, seja através da disponibilidade de informação ou da implementação de serviços de aplicativos. De acordo com o trabalho de (LIMA; BARBOSA, 2019), é possível observar que a área da enfermagem do Brasil está avançando em relação ao campo da tecnologia móvel, utilizando desse recurso para desenvolver aplicativos que são capazes de auxiliar a prática assistencial, a educação e a gestão em saúde. Um exemplo da vantagem da computação móvel é a implementação de um dispositivo portátil na área de saúde que é capaz de identificar as veias do paciente que são difíceis de enxergar a olho nu, via luz infravermelha, sem que seja necessário do profissional aplicar um torniquete para fazer com que as veias do paciente dilatem para ficarem visíveis, tornando o processo mais simples e com menos risco (PAN et al., 2019).

Portanto, nota-se que a humanidade depende muito das soluções móveis. Hoje, as pessoas vivem em função de seus dispositivos móveis (e.g., *smartphones*), que

disponibilizam uma infinidade de informações e recursos que fazem parte da vida do usuário. Um grande exemplo são as redes sociais que são muito utilizadas pelas pessoas para se conectarem umas com as outras, independente de suas localizações. Atualmente, qualquer pessoa é capaz de aprender ou se capacitar em qualquer área ou profissão, pois as informações necessárias estão todas disponíveis na Internet (i.e., seja em sites, cursos gratuitos ou pagos), além de existir aplicativos e sites que ajudam as pessoas a encontrarem emprego ou a buscarem utilizar um serviço (e.g., lanche, banco digital). Assim, fica claro que a computação móvel trouxe muitas oportunidades para as pessoas. Graças a computação móvel, a busca de oportunidades no mercado de trabalho, lazer, compras e vendas de produtos, comunicação social, e muito mais, tornaram-se executáveis remotamente, assim auxiliando no progresso da humanidade.

Contudo, é importante salientar que com a ampla disponibilidade de informações, é fundamental que os dados pessoais dos usuários sejam protegidos de maneira adequada, uma vez que, se não forem, podem ser acessados por terceiros mal intencionados, que podem furtar esses dados e comercializá-los ilegalmente. É necessário que as empresas estejam conscientes da importância da criptografia dos dados dos seus clientes, pois podem sofrer sanções caso não tomem as medidas adequadas de proteção. Além disso, com o aumento da popularidade dos dispositivos móveis, surgiram inúmeros aplicativos maliciosos que visam obter informações dos usuários e prejudicar os dispositivos. Por essa razão, é fundamental contar com uma barreira de proteção capaz de identificar esses *malwares* antes mesmo de serem instalados e executados. Isso será melhor abordado na seção 3.2.

3.2 Segurança da Informação e de Sistemas

A segurança da informação é um assunto que tem ganhado destaque no ambiente da computação. O termo segurança é usado com o significado de minimizar a vulnerabilidade de bens e recursos (SOARES; GUIDO; COLCHER, 1995). Os avanços tecnológicos têm possibilitado novas formas de acesso à informação, no entanto, isso amplia o risco de invasões aos dados pessoais. Infelizmente, existem pessoas mal-intencionadas que exploram as vulnerabilidades da segurança para interceptar informações confidenciais de suas vítimas. É fundamental que medidas de segurança sejam implementadas para garantir a proteção dos dados pessoais dos usuários (BINE, 2016).

A segurança da informação é a proteção da informação quanto a diversos tipos de ameaças as quais estão sujeitas. Para proteger os dados, ela segue alguns conceitos de segurança que são baseados em pilares. Os pilares são confidencialidade, integridade, disponibilidade, autenticidade e não-repúdio. A Tabela 1, a seguir, apresenta mais detalhadamente sobre cada um dos pilares, seguindo as informações dos trabalhos (HOEPERS, 2019) e (CABRAL, 2021):

Tabela 1 – Fundamentos da Segurança da Informação

Pilares	**Descrição**	**Ausência do pilar**
Confidencialidade	É a necessidade de garantir que as informações só possam ser acessadas por pessoas que tem autorização para tal.	A informação estará disponível para ser acessada por qualquer usuário, mesmo que não tenha permissão para isso.
Integridade	É a necessidade de garantir que as informações não serão alteradas por pessoas que não tem permissão, mantendo-as completas e corretas.	As informações podem ser alteradas antes do envio por um usuário malicioso, prejudicando o remetente.
Disponibilidade	É a necessidade de garantir que as informações estejam disponíveis para o usuário autenticado acessar quando for solicitado.	O usuário não conseguirá acessar a informação a partir de seu dispositivo.
Autenticidade	É a necessidade de garantir que o usuário que está acessando a informação é quem ele realmente diz ser.	Qualquer pessoa conseguirá acessar as informações sem precisar se autenticar no sistema.
Não-Repúdio	É a necessidade de garantir que o autor de uma ação em uma informação confirme seus atos e autorias.	Invasores poderão realizar atividades maliciosas que não serão descobertos, pois suas ações não serão identificadas.

Fonte: Autor (2022).

Atualmente, a grande maioria das pessoas está conectada na Internet através de dispositivos móveis, que ganhou popularidade a partir do século XX (BINE, 2016). Os dispositivos móveis são definidos como qualquer aparelho portátil que possui um sistema embarcado capaz de executar funções e processos, como navegar na Internet, reproduzir

mídia ou executar algum aplicativo, e podem variar desde notebooks, *smartphones* e tablets, até relógios de pulso, *smartwatches*, *smart speakers* e *E-readers*. Entretanto, esses dispositivos tornaram-se alvos frequentes de ataques por serem capazes de processar dados de forma equivalente ou superior a muitos computadores *desktops* e tendo alta probabilidade de conter informações confidenciais dos usuários (e.g., dados bancários e CPF). Consequentemente, a popularidade das plataformas de sistemas operacionais móveis cresceu, aumentando o número de aplicativos hospedados em suas estruturas. Em consequência, o número de aplicações maliciosas desenvolvidas para essas plataformas cresceu. Por exemplo, o sistema operacional Android é um dos mais utilizados hoje, sendo vitima de muitos ataques de desenvolvedores de aplicações maliciosas, segundo a revisão sistemática de literatura implementada por (SHARMA; RATTAN, 2021).

Por mais avançados que sejam os mecanismos de defesas, se não forem constantemente atualizados com tecnologias novas e dados atualizados que representem o comportamento da ameaça atual, não conseguirão detectar essas aplicações que estão em constante evolução, conforme os trabalhos (TAM, 2017) e (SAHAY, 2020) apresentam. Um exemplo recente (07/2022) de falha de segurança é o caso do *Google Play Protect* (GPP), o próprio mecanismo de defesa da *Google* integrado no Android, utilizado para detectar algum comportamento malicioso antes de instalar qualquer aplicativo no dispositivo móvel do usuário. O GPP não conseguiu identificar que haviam 36 aplicativos, na *Google Play Store*, que continham *adwares* e *malwares*, levando a mais de 10 milhões de *downloads* (SCHENDES, 2022). Sendo assim, conclui-se que o serviço do GPP ainda não é completamente eficiente (HUTCHINSON, 2019).

Um *malware* é basicamente um *software* que furta, modifica ou apaga as informações de um usuário sem o consentimento do mesmo. Existem diversos tipos de aplicativos maliciosos, cada um com sua finalidade. Alguns deles são:

- ***Ransomware*** - É um tipo de código malicioso que bloqueia o dispositivo da vitima, e muitas vezes criptografa os arquivos, solicitando a vitima que pague uma certa quantia para recuperá-los;
- **Trojan Bancário** - Esse tipo de *malware* tem por objetivo furtar as credenciais de plataformas bancárias online. Quando o aplicativo malicioso é instalado, ele executa diversas atividades no dispositivo da vítima, conseguindo furtar suas credenciais bancárias, onde são enviadas para o servidor do invasor. Muitas vezes eles conseguem burlar os sistemas de autenticação de múltiplo fator;
- **RATs *(Remote Access Trojans)*** - São trojans móveis que tem por objetivo espionar

o dispositivo da vítima. Esse tipo de *malware* pode realizar diversas ações no dispositivo infectado (e.g., gravar quais as teclas digitadas, gravar chamadas, tirar fotos, furtar credenciais de programas bancários, entre outros).

Os *malwares* citados acima são apenas alguns exemplos, pois existem diversos tipos de aplicações maliciosas, cada uma com uma funcionalidade específica. Esses *malwares* quando em ação, podem trazer muitos danos as vítimas. Segundo a publicação de (INSIDE, 2022), as principais informações do relatório de ameaças da ESET, são:

- O *ransomware* superou as piores expectativas em 2021, com ataques contra infraestrutura crítica e demandas de resgate com mais de US$ 5 bilhões somente no primeiro semestre de 2021;
- As detecções de *malware* bancário no Android aumentaram 428% em 2021 em comparação com 2020;
- Os números de ataques RDP (*Remote Desktop Protocol*) nas últimas semanas do terceiro trimestre de 2021 quebraram recordes anteriores, equivalentes a um crescimento anual de 897%.

Um *malware* pode ser detectado através da utilização de modelos de aprendizado de máquina, treinados com dados específicos para tal. Porém, existe uma técnica capaz de burlar o sistema de defesa de um modelo preditivo, que são os *poisoning attacks*. De acordo com um estudo sistemático (JAGIELSKI et al., 2018), *poisoning attacks* ou ataques de envenenamento de dados é uma forma de ataque que consiste em invadir o banco de dados de um sistema de defesa contra aplicações maliciosas e injetar dados falsos que enviesam o modelo preditivo, fazendo com que ele tome decisões erradas e abra vulnerabilidades em seu sistema, fazendo assim com que os cibercriminosos tenham total liberdade de invadi-los.

Além dos aplicativos maliciosos infiltrados nas plataformas de sistemas operacionais móveis e os ataques de envenenamento de banco de dados, existem outras técnicas para furtar as informações dos usuários, uma delas é o *phishing*. O *phishing* é um ataque que foca em furtar o dinheiro ou a identidade do usuário fazendo com que ele revele informações pessoais (e.g., número de cartão de crédito, dados bancários ou senhas de sites). Estima-se que em 2020 houve um crescimento de 124% de ataques a dispositivos móveis, onde os especialistas acreditavam que os *links* compartilhados via *WhatsApp*, e o trabalho remoto na época do Covid-19, tiveram grande parcela nisso, segundo (RODRIGUES, 2020). Foi quando o Covid-19 surgiu que os ataques

cresceram muito, fazendo as pessoas perceberem da importância da computação móvel e da segurança rígida que precisa ter nesses dispositivos. Outra técnica muito utilizada por cibercriminosos é a engenharia social, no qual usam a confiança de um usuário para persuadi-lo a fornecer seus dados importantes. Essa categoria de fraude pode acontecer dentro de uma empresa, desde que o criminoso consiga induzir um funcionário da mesma a realizar uma atividade que favoreça o início do ataque.

Com o crescimento desses ataques e violações de dados surge a LGPD, a Lei Geral de Proteção de Dados (13.709/2018), que entrou em vigor em 2020. A LGPD foi criada para garantir a privacidade de informações pessoais gerenciadas por empresas. Ela estabelece as responsabilidades dos coletores, armazenadores e tratadores de dados, e prevê medidas de segurança e punições para violações. A LGPD é aplicável a todas as empresas que possuam dados pessoais, incluindo informações de funcionários, clientes, parceiros e outros. Publicada em 14 de agosto de 2018, é aplicável a todas as empresas, de acordo com (ROMÃO, 2021). Segundo (VILELA, 2021), de acordo com o artigo 52, se a empresa violar as regras previstas da lei, ficará sujeita às seguintes sanções administrativas aplicáveis pela autoridade nacional:

1. Advertência, com indicação de prazo para adoção de medidas corretivas;
2. Multa simples, de até 2% (dois por cento) do faturamento da pessoa jurídica de direito privado, grupo ou conglomerado no Brasil, limitada, no total, a R$ 50.000.000,00 (cinquenta milhões de reais) por infração;
3. Bloqueio dos dados pessoais a que se refere a infração até a sua regularização ou a eliminação dos mesmos;
4. Suspensão parcial do funcionamento do banco de dados a que se refere a infração pelo período máximo de 6 (seis) meses, prorrogável por igual período, até a regularização da atividade de tratamento pelo controlador (incluído pela Lei nº 13.853, de 2019);
5. Suspensão ou a proibição parcial ou total do exercício de atividades relacionadas ao tratamento dos dados (incluído pela Lei n° 13.853, de 2019).

Segundo a publicação de (SANTIAGO, 2023), a primeira multa aplicada por violação da LGPD no Brasil ocorreu no dia 6 de Julho de 2023, marcando um marco significativo desde a entrada em vigor da legislação em setembro de 2020. A empresa *Telekall Infoservice*, atuante no setor de telefonia e com sede em Vila Velha (ES), foi alvo de duas sanções administrativas, totalizando R$ 14.400. A decisão foi tomada pela

ANPD (Autoridade Nacional de Proteção de Dados) após a conclusão de um processo administrativo iniciado em março de 2022. Durante a investigação, a empresa foi acusada de comercializar dados pessoais ao oferecer uma lista de contatos de WhatsApp de eleitores com o objetivo de disseminar material de campanha eleitoral em 2020. Além disso, a empresa não possuía um profissional responsável pelo tratamento adequado de dados pessoais, o que é obrigatório conforme previsto na legislação.

Após apresentar algumas técnicas de ataques utilizadas pelos cibercriminosos, faz-se necessário apresentar técnicas de segurança utilizadas para defender a integridade dos sistemas computacionais. Técnicas de segurança são um conjunto de medidas que são planejadas para diminuir o risco de que uma vulnerabilidade no sistema seja explorada por alguma pessoa mal intencionada, trazendo mais segurança para os dados do usuário. Uma das formas de detectar uma aplicação maliciosa é utilizando o aprendizado de máquina com análise estática ou dinâmica. A análise estática de um *malware* visa avaliar seu comportamento, extraindo características de seu código sem precisar executá-lo. A análise dinâmica visa monitorar o comportamento do *malware* durante sua execução, muitas vezes utilizando ferramentas que auxiliam na monitoração dos processos do sistema operacional e analisando as características do aplicativo. Essas características são os dados que fornecem a informação necessária para treinar modelos preditivos para detectar *malwares*.

Além das técnicas de aprendizado de máquina para detectar os *malwares*, é preciso ter boas práticas ao tratar os dados, por exemplo: manter um *backup* atualizado dos dados importantes do sistema, não utilizar as mesmas senhas, não clicar em qualquer *link* do email ou da Internet, não fornecer informações pessoais para qualquer pessoa, utilizar confirmação de múltiplos fatores, entre outras práticas que ajudam na redução do risco de acontecer alguma fraude. Essas são algumas boas práticas para lidar com dados em um dispositivo móvel, pois o mesmo além de ser muito utilizado pelo usuário e conter todas suas informações confidenciais, é um dos principais alvos de ataques por cibercriminosos.

3.3 Sistema Operacional Android

Um sistema operacional é um conjunto de processos executados pelo processador que tem como objetivo gerenciar os recursos do sistema da melhor forma possível e agir como interface entre o *hardware* e o usuário final. O Android é um sistema operacional para dispositivos móveis de código aberto (*open source*), baseado no sistema Linux,

sendo muito utilizado em *tablets* e *smartphones*. As principais funções de um sistema operacional são:

- **Gerenciamento de processos -** O sistema operacional gerencia os processos que serão executados no processador, permitindo que múltiplas aplicações sejam executadas ao mesmo tempo. Para isso é utilizado um escalonador que gerencia as rotinas da melhor maneira possível, levando em consideração a situação atual. No escalonamento de processos no Android, cada processo possui o seu intervalo de tempo para executar, e a cada encerramento de tempo de um processo ocorre a troca de contexto. Segundo (OLIVEIRA et al., 2014), o Android divide seus processos em duas classes de prioridade:
  - **Prioridade Dinâmica -** O escalonador realiza ajustes nas prioridades dando foco a utilização equilibrada do processador, isto é, o nível de prioridade dos processos podem ser atribuídos pelo tempo de sua execução. Se um processo em execução permanecer por um longo período, ele receberá um nível de prioridade menor do que um processo que está esperando na fila para ser executado pelo processador.
  - **Prioridade Estática -** Nessa classe, as prioridades são utilizadas por processos de tempo real, ou seja, quando não há mais processos em execução de tempo real, o escalonador libera espaço para executar os processos de prioridade dinâmica.

  O escalonador consegue diferenciar os processos de tempo real utilizando a política FIFO (First in, First out), dos processos de usuários escalonados utilizando a política Round Robin. Segundo (SANTOS, 2009), o escalonamento FIFO é um algoritmo não-preemptivo, pois nele os processos que estão prontos para serem executados são posicionados em uma fila organizada por ordem de chegada, onde cada processo, por sua vez, é colocado e mantido no processador até que seja totalmente executado. Já no escalonamento Round Robin, diferentemente da abordagem FIFO, os processos são organizados em uma fila com um tempo limite para suas execuções, denominado fatia de tempo (*time-slice*), onde toda vez que um processo é escolhido para ser executado pelo processador, uma nova fatia de tempo lhe é concedida. Se esse limite de tempo expirar, o sistema operacional irá interromper a execução do respectivo processo, salvará o contexto e direcionará-lo para o final da fila de pronto. Cabe enfatizar que todos os processos são carregados

e mantidos na memória e escalonados de acordo com o nível de prioridade.

- **Gerenciamento de memória -** É responsável por alocar espaço em memória para os processos que serão executados e liberar as posições de memória para os processos que já foram finalizados, assim o sistema garante que cada processo tenha seu próprio espaço na memória e que nenhuma aplicação utilize mais memória que a existente fisicamente (GOMES et al., 2012). Uma outra funcionalidade do gerenciador de memória é controlar o *swapping* de informação, na execução constante dos processos. De acordo com (JULIANA, 2017), o *swapping* é uma técnica focada em transferir todo o processo da memória principal para o disco (i.e., *swap out*) e vice-versa. Esta técnica é utilizada para auxiliar no problema da falta de memória durante a execução de alguns processos.
- **Sistema de arquivos -** Como a memória principal é volátil (i.e., todo o conteúdo é perdido quando a máquina é desligada), é necessário implementar algum método para armazenar e recuperar essas informações permanentemente. Os dados devem ser armazenados em um dispositivo periférico não volátil, como um disco rígido (HD), por exemplo, permitindo ser lido e gravado por processos (GOMES et al., 2012). Existem diversas opções de armazenamento de dados oferecidos pelo Android. Os trabalhos (COSTA, 2016) e (ANDROID, 2022) reforçam que é preciso escolher a opção de armazenamento mais adequada dependendo da necessidade do aplicativo em compartilhar dados e o espaço necessário para tal. Algumas das formas de armazenamento que podem ser utilizadas nos aplicativos, são:
    - Armazenamento Interno - Tem a capacidade de salvar arquivos diretamente na memória interna do dispositivo, os quais são privados pelo aplicativo que o criou para não poder ser acessado por nenhum outro. Quando o aplicativo é desinstalado do dispositivo, o arquivo privado criado por ele é excluído. Uma abordagem diferente seria utilizar a memória cache para armazenar os arquivos em diretórios temporários.
    - Armazenamento Externo - Os dispositivos móveis dão suporte para o armazenamento externo, isto é, uma memória externa que serve para aumentar a capacidade de armazenamento de dados do dispositivo, como um cartão de memória SD. Porém, todos os arquivos armazenados com a memória externa podem ser modificados pelos usuários e serem vistos por outras aplicações instaladas no dispositivo.

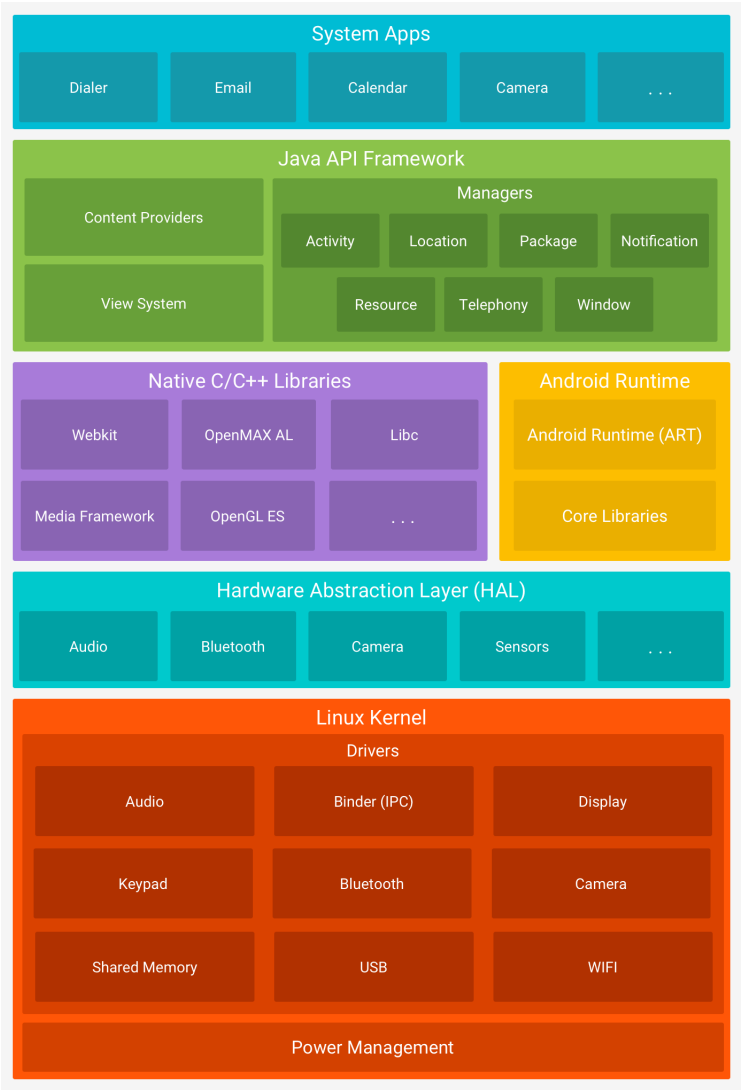
Com o passar dos anos o Android continuou evoluindo, sendo que cada versão exige mais processamento e memória disponível para serem instaladas do que as suas antecessores, pois com a evolução da tecnologia surgem mais recursos a serem gerenciados pelo sistema, abrindo assim mais possibilidades para os usuários. Por exemplo, com o lançamento do Android 13 pela Google, em agosto de 2022, a exigência para instalar a respectiva versão em dispositivos foi de pelo menos 16 GB de armazenamento interno e 2 GB de RAM. Dispositivos com menos de 16 GB de armazenamento interno estão fora do escopo da nova atualização, segundo (BICUDO, 2022).

- **Entrada e saída de dados -** Para que os dados sejam processados, é necessário com que o usuário interaja com a máquina e informe o que deve ser feito. Para isso são necessários dispositivos de entrada que recebam as informações vindas do usuário (i.e., externas a máquina), no caso de dispositivos como *smartphones* e *tablets*, a entrada de dados pode ser considerada a tela *touch*, que abrange tanto o teclado quanto o mouse, se comparado a computadores *desktops*. Para que os dados processados sejam visíveis ao usuário, é necessário alguma forma de transmiti-los para fora do dispositivo, ou seja, a saída de dados. Um exemplo é a própria tela do dispositivo, que serve como interface gráfica, exibindo todas as informações para o usuário. Outro exemplo são os fones de ouvidos, que processam o som que será emitido e transfere para os dispositivos auriculares. Todas essas ações são realizadas pelo sistema operacional.

Como já dito, o Android é uma pilha de *software* baseada em Linux. A Figura 3 ilustra grande parte dos elementos que compõe a arquitetura do Android.

O *kernel* ou núcelo é o principal componente do sistema operativo da maioria dos dispositivos computacionais, servindo como interface entre aplicativos e o processamento dos dados realizados a nível de *hardware*. O núcleo tem como função administrar os recursos do sistema da melhor forma possível. Observando a Figura 3, a base do Android é o *kernel* do Linux de código aberto, que está na camada inferior da pilha de *software*. Sua utilização permite que a plataforma aproveite os recursos de segurança principais (e.g., recursos para garantir a comunicação segura entre processos) e que seja possível fabricar *drivers* de *hardware* para um *kernel* conhecido. Segundo o trabalho de (SINGH, 2014), todo o sistema operacional é construído nesta camada, onde as seguintes funcionalidades são fornecidas: (a) Gestão de processos; (b) Gerenciamento de memória; e (c) Gerenciamento de dispositivos (e.g., câmera e teclado). O gerenciamento

Figura 3 – Arquitetura da plataforma Android.



Fonte: Site oficial do Android (2020)[2].

da memória virtual, rede, energia e *drivers* são da responsabilidade do *kernel* do Linux.

A camada seguinte é a de abstração de *hardware* (HAL), a qual fornece interface entre o *hardware* físico de um computador e o *software* que é executado nele mesmo. Sua função é disponibilizar uma plataforma estável para executar aplicações. Uma camada de abstração é uma forma de ocultar detalhes mais complexos para facilitar o entendimento sobre o sistema. A HAL é formada por módulos de bibliotecas que implementam uma interface para um determinado componente de *hardware*, como o *Bluetooth*, segundo (ANDROID, 2020).

A partir da versão 5.0 do Android (API nível 21) ou mais recente, cada aplicativo do dispositivo executa seu próprio processo com uma instância própria do *Android Runtime* (ART). O ART foi implementado para executar arquivos DEX com várias

máquinas virtuais em dispositivos de baixa memória, o qual é um recurso construído para consumir o mínimo de memória possível. Ele foi implementado para substituir a máquina virtual *Dalvik*, mudando a forma de como manipula os arquivos. O arquivo DEX *(Dalvik Executable)* é um formato de arquivo executável utilizado pelo sistema operacional Android para aplicativos *mobile*. Ele contém o código compilado de uma aplicação Android e é convertido a partir do código-fonte Java durante o processo de compilação de um aplicativo (ZHANG; BREITINGER; BAGGILI, 2016). O arquivo DEX é incluído no pacote APK (Android Package Kit), que é o formato de arquivo padrão para distribuição e instalação de aplicativos Android. O DEX é usado para garantir a compatibilidade com o sistema operacional Android e é essencial para a execução de aplicativos no dispositivo.

No momento em que um aplicativo está sendo instalado, o ART compilá-o usando uma ferramenta chamada *dex2oat*, o qual aceita arquivos DEX como entrada e gera um arquivo executável do aplicativo compilado. De acordo com (GHEDIN, 2014), o ART tornou-se superior na utilização do recurso "Coletor de lixo" do que seu antecessor *Dalvik*, no qual utiliza um tempo menor, consome menos memória e é mais eficiente neste trabalho. Lembrando que "Coletor de lixo" é um recurso que livra o desenvolvedor da responsabilidade de endereçar memória e liberá-la quando o aplicativo terminar de utilizá-la.

Em relação a camada que retrata as bibliotecas nativas em C/C++, diversos componentes e serviços do Android são implementados por códigos que utilizam funções de bibliotecas nativas das linguagens de programação C e C++ (e.g., ART e HAL), pois são linguagens bases. Para que as bibliotecas sejam utilizadas, a plataforma Android disponibiliza as *Java Framework APIs* para servirem como ponte entre as funcionalidades dessas bibliotecas e os aplicativos. Essas bibliotecas fornecem um conjunto de funcionalidades específicas que podem ser integradas a um projeto, facilitando e economizando tempo e esforço de um desenvolvedor.

As APIs implementadas na linguagem Java disponibilizam uma série de recursos do sistema operacional Android, as quais são compostas por classes e interfaces que estão divididas em pacotes. O site oficial do Android[3] informa que as APIs compõe blocos de programação essenciais para implementar aplicativos Android e administrá-los no dispositivo do usuário, necessitando de:

- Um sistema com uma interface adequada para programar a IU de um aplicativo;

[3]https://developer.android.com/guide/platform?hl=pt-br

- Um gerenciador de recursos;
- Um gerenciador de notificação;
- Um gerenciador de atividade para administrar o ciclo de vida dos aplicativos;
- Provedores de conteúdo que permitem que aplicativos acessem informações de outras aplicações.

O Android é acompanhado com um conjunto de aplicações nativas que já vem instaladas no dispositivo do usuário. Esses aplicativos fornecem funcionalidades básicas e essenciais para a experiência do usuário, por exemplo o envio de SMS, calendário, despertador, entre outros. Porém, como o Android é baseado em Linux de código aberto, é possível instalar outros aplicativos terceirizados que podem ser encontrados em plataformas como a Google Play Store[4].

A arquitetura do Android permite com que as aplicações possam ser executadas isoladamente uma das outras, tornando o sistema mais seguro. Por seu sistema ser de código aberto eles dão suporte a inúmeras aplicações desenvolvidas, o que traz uma diversidade de possibilidades de personalização do dispositivo do cliente. Segundo as informações levantadas do site Statista[5] pela equipe da Terra (ALMENARA, 2022), em agosto de 2022, o sistema operacional móvel do Google dominou cerca de 71,47% do mercado de *smartphones*, graças a sua estrutura ser baseada em Linux de código aberto que possibilitou a implementação e a distribuição do projeto sem precisar pagar taxas à Google pela sua divulgação. O Android tornou-se tão popular que existem mais de 24 mil dispositivos diferentes e mais de 2 milhões de aplicativos, de acordo com o seu site oficial (LOCKHEIMER, 2022). Ademais, foi anunciado que mais de 1 bilhão de usuários utilizam Android em todo o mundo.

Um pacote Android *(Android Application Pack)*, ou simplesmente APK, contém diversos elementos necessários para o funcionamento correto da aplicação. Um APK contém o código do aplicativo e diversos outros recursos que são utilizados na detecção de aplicações maliciosas utilizando análise estática ou dinâmica com aprendizado de máquina. Dentre esses elementos, destacam-se dois:

- **Classes.dex** - É o nome padrão do arquivo DEX *(Dalvik Executable)* em um aplicativo Android. É um arquivo executável que contém o código compilado da aplicação, convertido a partir do código-fonte Java. É incluído no pacote APK *(Android Package Kit)* e é essencial para a execução da aplicação no dispositivo

[4]https://play.google.com/store
[5]https://www.statista.com

Android. O formato DEX foi projetado para ser compatível com o sistema operacional Android e garantir o funcionamento eficiente da aplicação.

- **AndroidManifest.xml** - É um arquivo de manifesto do Android que descreve diversas informações do APK, como o nome, as versões de API, as permissões que dão direito de acesso a certas partes do sistema, entre outros. O *AndroidManifest.xml* é necessário para extração de grande parte das características estáticas.

Todo arquivo *AndroidManifest.xml* precisa conter:

- Nome do pacote do APK;
- Componentes do aplicativo, que são os provedores de conteúdo, serviços, broadcast receivers e atividades;
- Permissões que o aplicativo solicita para permitir acesso a determinadas áreas do sistema ou de outros aplicativos;
- Recursos de *hardware* e *software* exigidos pelo aplicativo, em outras palavras, requisitos mínimos requeridos da capacidade de um dispositivo para executar determinada aplicação.

As características do Android são muito importantes pois descrevem os aspectos do comportamento da aplicação, o que ajuda os algoritmos de aprendizado de máquina a detectarem aplicações maliciosas. Essas características podem ser permissões, atividades, intenções, provedores, receptores, serviços, chamadas de API, chamadas de Sistema, *Opcodes* (i.e., instruções de baixo nível em *assembly*), tráfego de rede, entre outras. As permissões são uma das características mais utilizadas para detectar *malwares* com o aprendizado de máquina, segundo a revisão sistemática de literatura implementada por (SHARMA; RATTAN, 2021). As permissões protegem o acesso à determinadas áreas do sistema que podem ser prejudiciais se acessadas por alguma pessoa mal intencionada, por isso ao baixar um aplicativo é muito importante verificar se as permissões que ele requer para funcionar condizem com o seu propósito (e.g., um aplicativo que tira foto requerer a permissão para acessar a câmera). Segundo (MODERNOS, 2015), as permissões podem ser classificadas em normal ou perigosa:

- **Normal -** Permissão que protege o acesso à chamadas de APIs que não apresentam riscos ao usuário (e.g., vibração do celular);
- **Perigosa -** Permissão que controla o acesso de chamadas de APIs que são potencialmente prejudiciais ao usuário (e.g., acesso a dados bancários);

As chamadas de API são métodos de classes externas a aplicação que descrevem o comportamento de uma determinada funcionalidade do sistema. É muito importante que os *datasets* contenham um grande número de amostras e uma diversidade de características que consigam descrever padrões de comportamentos de aplicativos maliciosos para detectarem as ameaças atuais. Porém, encontrar conjuntos de dados atualizados para Android é um problema, então a proposta da ADBuilder busca preencher essa lacuna.

A instalação de um aplicativo pode ser realizada através da plataforma oficial da *Google Play Store*, ou na busca de arquivos com extensão APK. Ao começar a instalação, o sistema de mecanismo de defesa integrado GPP (i.e., *Google Play Protect*) verifica se a aplicação é maliciosa, se não for, instala o aplicativo, senão, remove o arquivo e informa o usuário.

Para possuir todos os recursos que disponibiliza hoje, o Android passou por muitas modificações com o decorrer dos anos, cada uma deixando seu marco em uma nova versão. O nome das versões do Andoid seguem uma ordem alfabética, onde as duas primeiras versões receberam nomes famosos de robôs, e as versões seguintes passaram a ser nomeadas como doces. De acordo com a Tabela 2 é possível observar as últimas 11 (onze) versões do Android que são mais relevantes, apresentando suas principais características.

Tabela 2 – Histórico das versões do Android

Versão Android	Data	Características
Android 4.4 Kitkat	2013	Trouxe a inovação do *Google Now Launcher*, que permite os usuários realizarem suas tarefas e pesquisas através de voz, ativando o recurso apenas falando a frase: "Ok Google".
Android 5.0 Lollipop	2014	Trouxe a incorporação do Android em outros dispositivos como o GPS, os relógios *Smartwatches* e *Smart TVs*. Além disso, a barra de notificações foi integrada a tela de bloqueio, a segurança do dispositivo foi reforçada com bloqueio inteligente e foi desenvolvida uma lanterna nativa, de acordo com (LIMA, 2017; MEYER, 2020).
Android 6.0 Marshmallow	2015	Teve foco maior na segurança dos usuários, permitindo a eles darem permissão ao que os aplicativos poderiam ter acesso no dispositivo. Além disso, a vida da bateria foi aumentada, foi desenvolvido o suporte nativo de leitura de impressão digital e foi implementado um *backup* e restauração automática no Google Drive para dados e aplicativos, segundo (MEYER, 2020).
Android 7.0 Nougat	2016	Foi apresentado a funcionalidade de executar os aplicativos paralelamente (i.e., ao mesmo tempo), trazendo uma interface adequada para este recurso.
Android 8.0 e 8.1 Oreo	2017	Essas versões trouxeram o sistema de defesa *Google Play Protect*, que protege o dispositivo de aplicações maliciosas.
Android 9.0 Pie	2018	Essa versão foi dedicada a otimizar a experiência do usuário no sistema, melhorando a apresentação de notificações e acessibilidade aos recursos do sistema.
Android 10	2019	Essa versão trouxe melhorias no suporte de autenticação biométrica, suporte para protocolos de segurança *Wi-Fi*, suporte a Internet 5G e controle de localização com GPS.
Android 11	2020	De acordo com (VENTURA, 2020), essa versão trouxe consigo um recurso de conectar um dispositivo Android (i.e., com a versão 11) ao aplicativo *Android Auto* de um veículo compatível (i.e., suporte para executar a funcionalidade dessa versão). Além disso, essa versão trouxe uma grande inovação que é a funcionalidade de controlar dispositivos inteligentes que estão conectadas a uma casa, como tomadas e lâmpadas.
Android 12 e Android 13	2021 e 2022	Em relação as últimas duas versões lançadas, o foco foi na melhoria da privacidade e segurança, novos recursos (e.g., funcionalidade que faz os aplicativos hibernarem, ao não serem utilizados por um determinado tempo) e no aprimoramento da acessibilidade dos recursos do sistema para o usuário, segundo (MARQUES, 2021; ALECRIM, 2022).

Fonte: Autor (2022).

Desde o seu lançamento em 2008, o sistema operacional Android tem passado por várias atualizações e evoluções, conforme pode ser observado na Tabela 2, com a intenção de melhorar sua funcionalidade, segurança e usabilidade. Cada nova versão inclui melhorias no desempenho, novos recursos e correções de *software*. Além disso, as novas versões incluem aprimoramentos de segurança para proteger os dispositivos e aplicativos contra ameaças cibernéticas.

Ao longo dos anos, os cibercriminosos tem criado novos tipos de *malwares* para explorar vulnerabilidades no sistema operacional Android. Esses *malwares* incluem vírus, cavalos de Troia, *spywares* e *adwares*, entre outros. À medida que o sistema operacional Android evoluí, os cibercriminosos têm evoluído suas técnicas para contornar as medidas de segurança, como abordado na pesquisa (SAHAY, 2020). Isso faz com que seja crucial manter os dispositivos e aplicativos atualizados e utilizar ferramentas de segurança confiáveis para proteger contra ameaças cibernéticas.

3.4 Datasets e Aprendizado de Máquina

Os *datasets* são conjuntos de dados geralmente representados por uma matriz, onde cada linha representa uma amostra que será estudada e as colunas são as características que compõe seus dados (e.g., um *dataset* voltado para previsão do tempo teria suas linhas representando uma amostra do respectivo dia e suas colunas indicariam características como umidade do ar, velocidade do vento, precipitação, entre outros) (HOPPEN, 2018). Por exemplo, este trabalho tem como objetivo a segurança de dispositivos Android, para isso é necessário ter conjuntos de dados quantitativos e qualitativos para serem fornecidos aos modelos de aprendizado de máquina. Nesse caso, cada linha desse *dataset* seria um aplicativo Android diferente e as colunas seriam suas características (e.g., as permissões necessárias para o aplicativo ser executado, chamadas de API, versões de API, intenções, entre outros).

De acordo com o trabalho implementado por (DÜZGÜN et al., 2021), um *dataset* para detecção de *malwares* pode ser construído através de uma sistemática. Primeiramente, deve-se selecionar os repositórios, como VirusShare[6] e VirusSamples[7] para coletar as aplicações normais e maliciosas. Além disso, deve-se criar uma lista com os resumos criptográficos (e.g., SHA256 ou MD5) das aplicações que identificam-nas

[6]https://virusshare.com
[7]https://www.virussamples.com

unicamente, e enviá-los para o serviço online do VirusTotal. O serviço do VirusTotal irá validar se a aplicação é maliciosa e poderá informar a qual família pertence (e.g., vírus, *spyware*, *ransomware*, trojan) caso seja maliciosa. Após agrupar todos os *malwares* por família, é preciso utilizar alguma técnica ou ferramenta para extrair as características das aplicações, no trabalho foi utilizado um módulo em Python, conhecido como PEfile, para extrair as chamadas de API das aplicações. Ao final, com todos esses dados obtidos, apenas é preciso juntá-los no *dataset* e armazená-los em um arquivo com formato CSV. O *dataset* final gerado pelo trabalho de (DÜZGÜN et al., 2021) ficou com três tipos de características: resumos criptográficos, chamadas de API e tipo de *malware* (i.e., família).

Geralmente os *datasets* voltados para detecção de *malwares* Android são binários, i.e., representados por apenas zero (0) ou um (1), onde 1 indica que há a ocorrência daquela característica no aplicativo em questão, e 0 indica a não ocorrência. Um *dataset* muito popular e amplamente utilizado na literatura é o Drebin (ARP et al., 2014). Drebin é um *dataset* que contém 123.453 amostras de aplicativos normais e 5.560 *malwares*, contendo uma diversidade de características que incluem: permissões do sistema, chamadas de API, componentes (i.e., provedores, receptores, atividades e serviços), filtro de intenções e componentes de *hardware*, os quais foram extraídos através do arquivo AndroidManifest.xml e do arquivo DEX da aplicação, onde podem ser utilizados em análises estáticas. O Drebin é muito utilizado ainda hoje para treinar modelos de aprendizado de máquina para detectar *malwares*, para levantar estudos sobre as diversas técnicas de detecção ou até mesmo realizar experimentos, como visto na literatura (VILANOVA et al., 2021).

Aprendizado de máquina é um campo da inteligência artificial que se concentra em desenvolver algoritmos e modelos computacionais que podem aprender com dados e tomar decisões sem serem explicitamente programados. Existem vários tipos diferentes de aprendizado de máquina, incluindo aprendizado supervisionado, não supervisionado e por reforço. Aprendizado de Máquina é um campo da inteligência artificial que se concentra em criar algoritmos e modelos que podem aprender e melhorar a partir de dados.

De acordo com o trabalho (LIU et al., 2020), o aprendizado de máquina é dividido em três categorias principais: aprendizado supervisionado, aprendizado não supervisionado e aprendizado por reforço. O aprendizado supervisionado é o tipo de aprendizado de máquina onde o algoritmo é treinado com um conjunto de dados rotulados (conhecido como *dataset* de treinamento), que possui entradas e saídas esperadas. O

objetivo é que o algoritmo aprenda a mapear as entradas para as saídas corretas, de forma a poder fazer previsões precisas sobre novos dados. Exemplos de tarefas de aprendizado supervisionado incluem classificação, regressão, entre outros. Aprendizado não supervisionado, por outro lado, é o tipo de aprendizado de máquina onde o algoritmo é treinado com um conjunto de dados não rotulado (ou seja, sem saídas esperadas). O objetivo é que o algoritmo encontre padrões e estruturas nos dados por conta própria. Exemplos de tarefas de aprendizado não supervisionado incluem agrupamento, redução de dimensionalidade e detecção de anomalias. Aprendizado por reforço é um tipo de aprendizado de máquina onde o algoritmo é treinado por meio de uma série de tentativas e erros, onde ele é "recompensado" ou "punido" de acordo com seu desempenho. O objetivo é que o algoritmo aprenda a tomar decisões e realizar ações que maximizem a recompensa. Exemplos de tarefas de aprendizado por reforço incluem jogos de computador, robótica e outras tarefas de tomada de decisão.

A eficiência do aprendizado de um modelo de inteligência artificial vem do conjunto de dados que lhe é fornecido. Para construir um *dataset* de qualidade contendo dados com alto grau de informação, faz-se necessário levantar quais as características que precisam ser trabalhadas e inclusas no mesmo. As características são aspectos que descrevem uma aplicação, podendo variar desde permissões necessárias para o aplicativo acessar certas funcionalidades do dispositivo até os metadados gerais presentes no APK. Algumas das características de um aplicativo Android que podem ser inclusas em um *dataset* são:

- **Funções Hash** - São sequências de caracteres que identificam unicamente um aplicativo Android, por exemplo o SHA256.
- **Pacote** - É o pacote do aplicativo Android.
- **Permissões** - Permissões necessárias para que o aplicativo execute alguma ação ou obtenha acesso a algumas informações e funcionalidades do dispositivo do usuário. As permissões são uma das características mais utilizadas em estudos e pesquisas de detecção de *malwares* Android, como (WANG et al., 2019b), (PAN et al., 2020), (SHARMA; RATTAN, 2021) e (VILANOVA et al., 2021).
- **Intenções** - É um objeto de mensagem que pode ser usado para solicitar uma ação de outro componente do aplicativo.
- **Serviços** - É um componente executado em segundo plano para realizar operações de execução longa ou trabalho para processos remotos.

- **Receptores** - É um componente que faz o sistema entregar eventos ao aplicativo. Podem ser usados como um sistema de troca de mensagens entre aplicativos.
- **Provedores** - São componentes que gerenciam um conjunto de dados compartilhado do aplicativo, que podem armazenar nos sistemas de arquivos, em banco de dados SQLite, na Web ou em qualquer local de armazenamento persistente acessível ao seu aplicativo.
- **Atividades** - É um componente que serve como ponto de entrada para a interação com o usuário.
- ***Opcodes*** - São a referência à instrução que um determinado processador possui para conseguir realizar determinadas tarefas. Em outras palavras, são instruções de baixo nível em *assembly*.
- **Chamadas de API** - São a referência do método de uma classe, contida dentro do pacote da API, no aplicativo, chamada para executar alguma funcionalidade. Elas são uma forma de fazer programas e aplicativos conversarem e trocarem informações entre si. Além disso, descrevem o comportamento do APK.
- **Strings** - São sequências de caracteres que são importantes para identificar um comportamento malicioso, por exemplo URLs que o aplicativo utiliza.
- **Tráfego de rede** - São dados que são enviados de uma rede de origem até uma rede destino em um determinado intervalo de tempo, onde os dados são encapsulados em um pacote que é a unidade essencial o qual é utilizado para a transferência de arquivos via rede.
- **Chamadas de Sistema** - São funções utilizadas pelos aplicativos para solicitarem alguma execução ao serviço do *kernel* do sistema operacional, e podem ser vistas como rotinas que permitem o aplicativo solicitar uma ação que requere um privilégio especial do sistema.
- **Informações Gerais do Sistema** - Informações como consumo de RAM, uso da bateria, entre outros.

A qualidade das características afeta diretamente a capacidade do modelo de aprendizado de máquina de generalizar para novos conjuntos de dados. Isso significa que se as características são escolhidas de forma inadequada ou não representativas, o modelo pode apresentar baixa precisão quando é usado com novos dados. A quantidade de características é importante, pois se houver muitas características, o modelo pode sofrer de *overfitting*, ou seja, ele pode se ajustar muito bem aos dados de treinamento,

mas não generalizar para novos dados. A diversidade das características é importante, pois se elas são muito semelhantes, o modelo pode ter dificuldade para diferenciar entre elas e, portanto, pode não aprender adequadamente. A atualidade das características é importante, pois se elas são desatualizadas, o modelo pode não ser capaz de se adaptar a mudanças nos dados. Por fim, a precisão dos dados e a qualidade dos dados é importante, pois se os dados contêm erros ou *outliers*, isso pode afetar negativamente o desempenho do modelo. Em resumo, a escolha adequada das características e a qualidade dos dados são fundamentais para garantir que o modelo de aprendizado de máquina seja preciso e capaz de generalizar para novos conjuntos de dados.

3.5 Trabalhos Correlatos

Esta seção é destinado a apresentar trabalhos correlatos que foram avaliados e estudados, a fim de identificar suas características e, desta forma, propor a implementação da ADBuilder.

O trabalho apresentado por (LASHKARI et al., 2018) propõe uma abordagem sistemática para gerar um conjunto de dados de *malwares* do sistema Android usando *smartphones* reais. Primeiramente, o trabalho passa por uma etapa de coleta de dados, realizando pesquisas sobre os *datasets* da área de detecção de *malwares* Android disponíveis, e busca de repositórios de APKs. Foram realizadas coletas de aplicativos maliciosos, buscando abranger diversas famílias de *malwares*. Em seguida, foi realizada uma análise das amostras coletadas (i.e., *malwares*), onde foram submetidos ao serviço do VirusTotal (i.e., um serviço online que rotula um arquivo como malicioso ou não) para garantir que as aplicações são maliciosas. Um dos pontos mais importantes discutidos neste trabalho é o fato do problema da defasagem na rotulagem nos *datasets* ser comum, conjuntos de dados esses que foram criados e nunca mais foram atualizados, podendo apresentar informações erradas da rotulação da amostra (e.g., um aplicativo ser considerado benigno no *dataset* enquanto ele na verdade é malicioso). Isso se dá pela evolução das aplicações e dos antivírus, conforme abordado nos trabalhos (TAM, 2017) e (SAHAY, 2020). Ainda neste trabalho, foi realizado uma descrição sobre as características dos aplicativos Android (i.e., permissões, chamadas de API, chamadas de sistema, tráfego de rede, entre outros). Após essa descrição, foi realizada a etapa de extração de características, onde foram utilizadas duas ferramentas: *Appmon*[8], que

[8]https://github.com/dpnishant/appmon

extrai as características estáticas dos aplicativos (e.g., permissões e chamadas de API); e *CICFlowMeter*[9], que extrai as características dinâmicas das aplicações (e.g., chamadas de Sistema e tráfego de rede).

A pesquisa desenvolvida por (DHALARIA et al., 2021) traz uma proposta de uma abordagem mais eficiente para detectar *malwares* no Android, juntando análises estáticas e dinâmicas. Este trabalho também passa por etapas, como: (a) coleta de dados, selecionando os repositórios *VirusShare*, *Apkpure* e *Apkmirror* para obtenção dos aplicativos Android, tanto benignos quanto malignos; (b) preparação dos dados, etapa necessária para adequar os dados do *dataset* para o aprendizado de máquina, onde foram removidas amostras duplicadas e problemas na rotulação; e (c) extração de características, onde foi desenvolvido um código próprio dos autores para extrair as características estáticas, e foi utilizado a ferramenta *CuckooDroid*[10] que extrai as características dinâmicas das aplicações.

O trabalho implementado por (LI et al., 2017) propõe a construção de um conjunto de dados significativos de aplicativos Android, com cerca de 5 milhões de amostras e 20 tipos de metadados diferentes. Os metadados consistem em: tamanho em bytes dos APKs, resumos criptográficos (e.g., SHA256), nome, pacote, origem do APK, entre outros. Outros dois metadados que estão inclusos são a informação da quantidade de *scanners* do VirusTotal que identificaram o APK como malicioso, e a respectiva data da análise do mesmo, o que facilita para os pesquisadores encontrarem um limiar que melhor se adéque as suas pesquisas. Além da rotulação, o trabalho também informa de onde os dados foram coletados e sobre a extração de características realizada a partir do arquivo manifesto do aplicativo. Porém, não foi abordado como o *dataset* foi realmente construído.

Outra pesquisa que tem como objetivo construir um conjunto de dados é o trabalho realizado por (WANG et al., 2019a), que coleta os aplicativos através da Google Play Store. Assim como um dos objetivos da ADBuilder, este trabalho realiza um levantamento de *malwares* recentes, que passaram despercebidos pelo mecanismo de defesa da Google Play, tornando-os amostras significativas que representam um comportamento de possíveis ameaças em potencial. Este trabalho utiliza o serviço do VirusTotal para rotular os aplicativos que foram coletados e analisar seus comportamentos através da extração de características realizada por uma API da Google. Ao final do trabalho, foi construído um *dataset* contendo 9133 amostras maliciosas de 56 famílias de *malwares* diferentes. As famílias foram identificadas através de uma API implementada

[9]https://github.com/ahlashkari/CICFlowMeter
[10]https://github.com/idanr1986/cuckoodroid-2.0

em Python, chamada AV Class[11].

A pesquisa realizada por (CATAK; YAZI, 2019) consiste em analisar 7107 aplicativos maliciosos diferentes através de diversas famílias de *malwares*, que são transformados em dados formatados para serem utilizados por algoritmos de classificação de aprendizado de máquina. Este trabalho informa de onde os dados foram coletados, explica sobre a rotulação utilizada pelo serviço do VirusTotal, e aborda o ferramental para a extração de características, na qual se baseia em uma análise dinâmica utilizando a ferramenta Cuckoo Sandbox[12]. Além disso, o trabalho aborda o processo da construção do *dataset*, porém não retrata sobre a limpeza dos dados e nem da validação das características extraídas.

O trabalho implementado por (DÜZGÜN et al., 2021) discute sobre os diferentes tipos de análises (i.e., estáticas, dinâmicas e híbridas), além de abordarem sobre a importância da necessidade de um conjunto de dados recente e atualizado para ajudar os modelos de aprendizado de máquina a detectarem as aplicações maliciosas que vem crescendo muito ultimamente. Além disso, o trabalho analisa o desempenho da classificação dos *malwares* utilizando um *dataset* balanceado (i.e., mesma quantidade de aplicativos maliciosos e normais) e desbalanceado, o que gera diferença na decisão do modelo, podendo até enviesá-lo. Eles coletam as amostras de *malwares* dos repositórios VirusShare e VirusSample. Ao final, os pesquisadores utilizam algoritmos de aprendizado de máquina como Random Forest, Support Vector Machine e XGBoost para treiná-los com os dados obtidos, com classificação dinâmica, chegando em uma acurácia de 94% na detecção dessas aplicações maliciosas.

A pesquisa realizada por Mahindru (MAHINDRU et al., 2020), denominada SOMDROID, propõe uma abordagem de classificação de *malwares* utilizando permissões e chamadas de API extraídas dos aplicativos Android, onde utiliza a técnica de aprendizado de máquina não supervisionado. O aprendizado não supervisionado baseia-se em treinar um algoritmo com dados que não estão rotulados. Sendo assim, os algoritmos irão buscar descobrir padrões ocultos para agrupar um conjunto de informações com sua semelhanças ou diferenças, por exemplo. O SOMDROID realiza uma seleção das características mais significativas através de 6 (seis) abordagens diferentes de classificação de recursos. Com os recursos selecionados, os pesquisadores implementaram o algoritmo Self-Organizing Map (SOM) de Konohen e calcularam 4 (quatro) parâmetros de desempenho distintos. Ao final, o trabalho mostrou-se promissor,

[11]https://github.com/malicialab/avclass
[12]https://cuckoosandbox.org

revelando que a estrutura do SOMDROID é capaz de detectar *malwares* com uma acurácia de 98,7%.

Outro trabalho desenvolvido por Mahindru (MAHINDRU et al., 2021), chamado de MLDroid, propõe um estrutura baseada na Web que auxilia na detecção de aplicações maliciosas para dispositivos Android. Para realizar a detecção, o MLDroid realiza uma análise dinâmica das características, analisando o comportamento da aplicação enquanto ela está sendo executada, através de uma Sandbox. Assim como o trabalho do SOMDROID (MAHINDRU et al., 2020), o MLDroid realiza seleção dos recursos mais significativos para treinar os modelos de aprendizado de máquina. O experimento foi realizado em mais de 500.000 aplicativos Android. Foram utilizados 4 (quatro) algoritmos distintos de aprendizado de máquina paralelamente, sendo eles um algoritmo de aprendizado profundo, primeiro agrupamento mais distante, Y-MLP e Floresta de Árvore de Decisão de Conjunto Linear. Com isso, os resultados do desempenho da taxa de detecção foi de 98,8% para detectar *malwares* Android.

Ainda, (ROTONDO, 2016) propõe a implementação de uma ferramenta (PROSEC) para identificar e monitorar processos maliciosos ou desconhecidos no sistema Android, buscando garantir a segurança dos dispositivos móveis e usuários. De acordo com o estudo: “o intuito é manter o usuário informado sobre todas as aplicações ativas e incrementar a segurança no dispositivo”. Para a realização do projeto foi preciso estruturar a metodologia em etapas, nas quais abrangiam: o levantamento do referencial teórico que auxiliaria no entendimento do problema e na construção da solução; implementação de um módulo; o levantamento dos requisitos da aplicação; e a implementação e realização de testes da ferramenta. O módulo da ferramenta é o filtro de segurança, no qual é responsável por armazenar e analisar os processos que estão em execução no sistema. Após a etapa de implementação, foram realizados os testes para garantir que a ferramenta cumpra com as especificações. Com isso, ao final do trabalho a ferramenta é capaz de identificar um comportamento malicioso, incrementando a segurança do dispositivo móvel Android.

Tabela 3 – Comparação entre ADBuilder e trabalhos relacionados.

Trabalho	Coleta de Dados	Preparação dos Dados	Rotulação	Extração de Características	Consolidação das Características	Construção do *dataset*
(LASHKARI et al., 2018)	✓	✗	✓	✓	✗	✓
(LI et al., 2017)	✓	✗	✓	✓	✗	✗
(WANG et al., 2019a)	✓	✗	✓	✓	✗	✗
(CATAK; YAZI, 2019)	✓	✗	✓	✓	✗	✓
(DÜZGÜN et al., 2021)	✓	✗	✓	✓	✗	✓
(DHALARIA et al., 2021)	✓	✓	✓	✓	✗	✗
(MAHINDRU et al., 2020)	✓	✗	✓	✓	✗	✓
(MAHINDRU et al., 2021)	✓	✗	✓	✓	✗	✓
(ROTONDO, 2016)	✗	✗	✗	✗	✗	✗
ADBuilder	✓	✓	✓	✓	✓	✓

Fonte: Autor (2022)

Na Tabela 3 é apresentado um resumo dos principais trabalhos sobre processos e ferramental para a construção de *datasets* Android. Exceto a proposta da ADBuilder, nenhum dos demais trabalhos é reprodutível e nenhum deles disponibiliza o código de instrumentação do processo de construção do *dataset*. Em termos de reprodutibilidade, os trabalhos falham em aspectos como abordar detalhadamente cada etapa do seu processo, disponibilizar o ferramental e informar o limiar utilizado para rotular um *malware*. Por exemplo, o trabalho (LI et al., 2017) não descreve como foi a etapa de limpeza de dados, como o *dataset* foi construído e nem qual o ferramental utilizado para isso, o que inviabiliza a sua reprodução.

Todos os trabalhos mapeados (com exceção do trabalho implementado por (ROTONDO, 2016) que não tem como objetivo construir um *dataset* ou coletar amostras de aplicativos Android) informaram suas fontes de dados, isto é, de onde foram coletadas as amostras. Entretanto, é importante observar que os trabalhos, apesar de recentes, não incluem repositórios como o AndroZoo e não possuem como meta construir *datasets* significativos, isto é, com um número estatisticamente representativo de amostras e características considerando a população total de aplicativos Android dos mais variados repositórios e mercados. Além disso, nenhum dos trabalhos demonstrou preocupação com a catalogação de amostras de *malwares* sofisticados, como os que são periodicamente identificados e removidos manualmente dos repositórios, apenas o trabalho de (WANG et al., 2019a) realizou um levantamento sobre os *malwares* que passaram despercebidos pela Google Play.

Tabela 4 – Ferramental dos Trabalhos Relacionados.

Trabalho	**Repositório de Aplicativos**	**Rotulação**	**Extração de Características**
(LASHKARI et al., 2018)	Google Play Store, VirusTotal, Contagio	VirusTotal (Benignos menos de 2)	Appmon, CICFlowMeter
(LI et al., 2017)	Google Play Store, Anzhi, App China, F-Droid	VirusTotal	Android Manifest
(WANG et al., 2019a)	Google Play Store	VirusTotal (mais de 20), AVClass (Família do)	API da Google Play
(DÜZGÜN et al., 2021)	VirusTotal, VirusShare, VirusSamples	VirusTotal	Framework Python PEfile
(DHALARIA et al., 2021)	VirusShare, Apkpure, Apkmirror	Avira Antivirus	(Estático) Baksmali Diassembler, AXMLPrinter2; (Dinâmico) CuckooDroid
(CATAK; YAZI, 2019)	Github	Cuckoo Sandbox, VirusTotal	Cuckoo Sandbox
(MAHINDRU et al., 2020)	Google Play Store, APKmirror, AllFreeAPK, AppChina, HiApk, Mumayi, SlideMe, PandaApp, Botnet samples, MalwareGenomeProject, AndroMalShare	VirusTotal, Microsoft Windows Defender	Não detalham e não disponibilizam código
(MAHINDRU et al., 2021)	Google Play Store, HiAPK, AppChina, Mumayi, Gfan, SlideMe, PandaApp, AndroMalShare, MalwareGenomeProject	VirusTotal, Microsoft Windows Defender	Não detalham e não disponibilizam código
(ROTONDO, 2016)	-	-	-
ADBuilder	AndroZoo	VirusTotal	AndroGuard

Fonte: Autor (2022)

Com relação à rotulagem, a maioria dos trabalhos utiliza o VirusTotal, como pode ser observado pela Tabela 4. Entretanto, apenas os trabalhos (LASHKARI et al., 2018) e (WANG et al., 2019a) informam o limiar utilizado (2 e 20 *scanners*, respectivamente), mesmo que um valor subjetivo (i.e., sem análise de dados para suportar a escolha), utilizado para rotular o aplicativo (i.e., entre benigno ou maligno). O trabalho implementado por (WANG et al., 2019a) considera um aplicativo malicioso se no mínimo possui 20 *scanners* positivos indicando-o como tal. Porém, para rotular os benignos eles confiam na Google Play Store, e.g., se o APK em questão foi rotulado como *malware* no VirusTotal, porém ele continua na Google Play Store, eles o consideram como uma aplicação normal. Já o trabalho (LASHKARI et al., 2018) apenas informa que os aplicativos são considerados benignos se possuem menos que 2 *scanners* positivos, não informando nada sobre a rotulação dos *malwares*. Os demais trabalhos não fornecem informações sobre a rotulagem propriamente dita, isto é, não informam se utilizam um limiar (e.g., 1 *scanner* positivo) ou se incluem a informação de quantidade de *scanners* no *dataset* para análise.

Nenhum dos trabalhos (LI et al., 2017), (LASHKARI et al., 2018) e (WANG et

al., 2019a) abordam uma etapa de preparação dos dados, a qual é muito importante para adequar os dados de um *dataset* para ser utilizado por um algoritmo de aprendizado de máquina. Sem essas informações não é possível reproduzir o trabalho. Como pode ser observado na Tabela 4, em relação a etapa de extração de características, nenhum dos trabalhos voltados para análise estática utilizou a ferramenta Androguard[13], considerada a mais completa e eficaz para a extração de características estáticas, conforme visto na literatura (PONTES et al., 2021). Além disso, nenhum dos trabalhos realiza uma etapa de consolidação das características extraídas, validando se estão de acordo com outros serviços, ferramentas ou *datasets*. De todos os trabalhos, apenas o implementado por (LASHKARI et al., 2018) abordou sobre o processo da construção do seu conjunto de dados.

O trabalho de (LASHKARI et al., 2018), além de compartilhar do mesmo objetivo do presente estudo, que é construir *datasets* voltado a detecção de *malwares* para Android, ele realiza etapas semelhantes que serão implementadas e incorporadas na ferramenta ADBuilder, proposta neste trabalho, isto é, a extração de características e a rotulação dos aplicativos (i.e., entre benigno ou maligno). Para a rotulação dos aplicativos é utilizado o serviço do VirusTotal, o qual é um forte candidato a ser incorporado como serviço do módulo de rotulação da ADBuilder. Outros dois processos semelhantes que serão executados durante esta pesquisa serão o estudo sobre as características do Android e na utilização de uma ferramenta de extração de características estáticas dos APKs, para ser possível implementar o módulo de extração.

De acordo com a Tabela 3, o trabalho de (DHALARIA et al., 2021) além de possuir muitas etapas semelhantes a este trabalho e ao (LASHKARI et al., 2018) (i.e., coleta de dados, extração de características e rotulação dos aplicativos), realiza um processo muito importante para os *datasets* que é a etapa de preparação dos dados. Esta etapa é baseada na limpeza e adequação dos dados do *dataset* para o aprendizado de máquina do modelo preditivo, sendo uma etapa necessária e que será implementada e incorporada na ferramenta ADBuilder. Por fim, o trabalho (ROTONDO, 2016) propõe uma ferramenta (PROSEC) para solucionar um problema específico no sistema operacional Android, que é detectar processos maliciosos causados por *malwares*. Por mais que a proposta dessa ferramenta e da ADBuilder sejam diferentes, elas compartilham um objetivo importante em comum que é ampliar a segurança nos dispositivos do sistema Android, a PROSEC buscando detectar qualquer processo malicioso no sistema, e a

[13]https://github.com/androguard/androguard

ADBuilder buscando construir conjuntos de dados atualizados que possam ser utilizados no treinamento de modelos preditivos que detectem aplicações Android maliciosas.

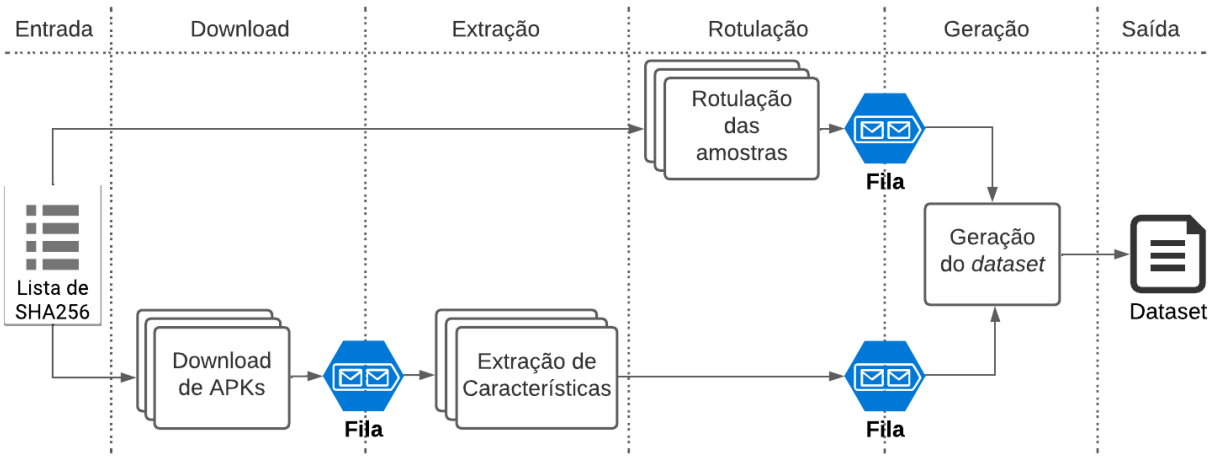
4 ADBUILDER - UMA FERRAMENTA PARA CONSTRUÇÃO DE *DATASETS* ANDROID

O presente Capítulo tem por finalidade apresentar a proposta da ferramenta para a construção de *datasets* deslumbrando questões de segurança do Android que outras ferramentas, como mostrado nos trabalhos correlatos, não apresentam. Desta forma, a seção 4.1 apresentará a descrição da proposta da ferramenta ADBuilder com sua arquitetura. A seção 4.2 apresentará a validação dessa proposta. A seção 4.3 apresentará a modelagem baseada em requisitos da engenharia de *software*. A seção 4.4 abordará sobre o estudo das tecnologias que foram integradas na ferramenta. Na seção 4.5 serão descritos os requisitos de execução. A seção 4.6 descreverá a implementação da ADBuilder. E por fim, a seção 4.7 apresentará os experimentos realizados, a validação da ferramenta e os resultados obtidos.

4.1 Descrição da Proposta

Dado a dificuldade existente em encontrar um *dataset* de qualidade com dados atualizados para a detecção de *malwares* no Android, que apresente uma grande diversidade de características e que seja quantitativo em termos de amostras, esse trabalho teve como objetivo propor desenvolver uma ferramenta integrada e automatizada capaz de construir conjuntos de dados atualizados que sejam eficazes para o treinamento de modelos preditivos de aprendizado de máquina. A ferramenta foi composta por 4 (quatro) módulos independentes que implementam 6 (seis) etapas bem definidas: (a) catalogação e (b) rotulação dos aplicativos, (c) extração e (d) consolidação das características, (e) criação do *dataset*, e seu (f) pré-processamento. O módulo de *download* foi responsável por baixar os aplicativos. Já o módulo de extração foi responsável por extrair suas características. O módulo de rotulação realizou a rotulagem das amostras (i.e., indicando se o aplicativo era malicioso ou não). Por fim, o módulo de geração foi o responsável por construir o *dataset*, sendo necessários os dados tanto de rotulação quanto de extração do APK para serem adicionados ao conjunto de dados final. Na Figura 4 é apresentada a proposta da arquitetura da ferramenta ADBuilder.

Figura 4 – Arquitetura da ADBuilder.

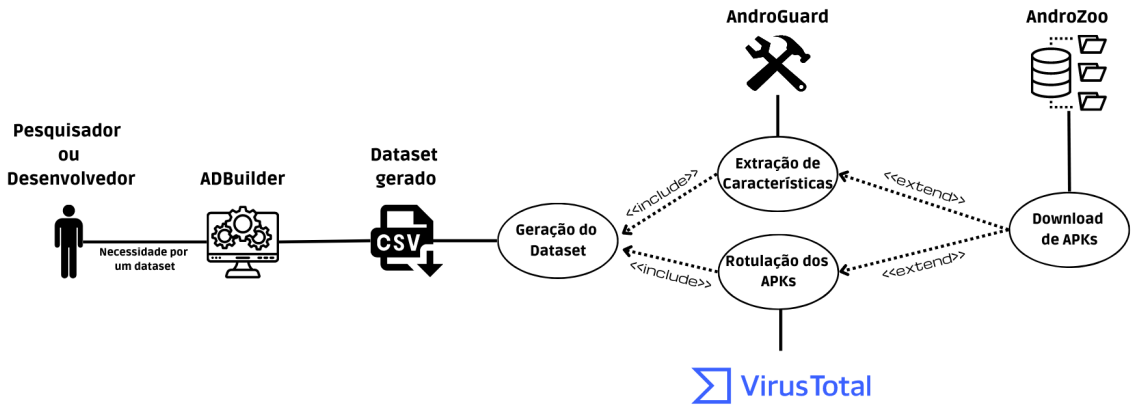


Fonte: Autor (2022).

É possível observar na Figura 4 que os módulos são independentes, sendo fracamente acoplados entre si. Eles se comunicam através de filas, seguindo o princípio de produtor-consumidor. Para cada fila há um módulo que produz novos dados (e.g., *download* e rotulação) e outro módulo que consome os dados da fila (e.g., geração). Vale ressaltar que o módulo de extração atua tanto como consumidor como produtor, consumindo dados que vem do *download* e produzindo dados para a fila de geração. Além disso, os módulos de *download* e extração são etapas que podem ser executadas por múltiplos processos, o que permitiu o aceleramento da construção do *dataset*.

Cada módulo da ADBuilder foi implementado com ajuda de ferramentas que desempenham papéis importantes. De acordo com a Figura 5, é possível observar que no primeiro módulo (i.e., *download*) foi utilizado o repositório AndroZoo para coletar as amostras. No segundo módulo (i.e., extração) foi utilizado a ferramenta Androguard para extrair as características. O terceiro módulo (i.e., rotulação) foi implementado com a API do VirusTotal para rotular os aplicativos. Por fim, o último módulo (i.e., geração) baseou-se em juntar os dados do segundo e terceiro módulos para construir o *dataset*.

Figura 5 – Introdução dos Módulos.



Fonte: Autor (2022)

A proposta da ADBuilder consistiu em aplicar algumas etapas importantes na construção do *dataset*, como a preparação dos dados, que consiste em retirar os ruídos causados por dados sujos que reduzem a qualidade do *dataset*.

Elencou-se mais de 100 *malwares*[14] Android recentes que passaram despercebidos pelo mecanismo de defesa da GPP e se infiltraram na plataforma da Google Play Store, infectando milhões de dispositivos. A coleta e adição desses *malwares* recentes no conjunto de dados será uma ajuda na formação dos modelos e será uma informação valiosa para identificar essas ameaças maliciosas. O objetivo é melhorar a precisão e a eficiência na detecção de ameaças para proteger os usuários e dispositivos contra ataques maliciosos. A inserção desses *malwares* recentes e não detectados no *dataset*, fornecerá uma base de dados mais abrangente e atualizada para treinar e testar modelos de aprendizado de máquina. Os *malwares* coletados estão disponíveis em um repositório Github.

[14]https://docs.google.com/spreadsheets/d/1fH2vBx1n0zS0Va5GxUeXi-cM0iiLrZNf/edit#gid=1343249897

4.2 Validação da Solução

Observa-se na Figura 4 que a arquitetura da ferramenta foi proposta com o objetivo de automatizar o processo de construção de *datasets* para detecção de *malwares* no Android. Cada módulo foi desenvolvido com o uso de conhecimentos específicos e ferramentas para garantir a eficácia e eficiência da ferramenta. Logo após isso, os estudos foram realizados para garantir a integração das ferramentas e a validade da execução da ADBuilder.

O código da funcionalidade dos dois primeiros módulos (i.e., *download* e extração) foram implementadas. Para validar essa implementação, as ferramentas de cada módulo implementado foram testadas individualmente para garantir a correta execução da funcionalidade que foi integrada. Por exemplo, o primeiro módulo baixa os aplicativos Android, obtidos através do repositório AndroZoo. Este repositório contém um vasto *dataset* de metadados de milhões de aplicativos que podem ser selecionados e coletados baseados em critérios específicos. A API do AndroZoo permite acelerar o processo de coleta dos APKs através de paralelismo. Foi utilizado o comando *curl*, fornecendo uma chave de API e um SHA256 (resumo criptográfico que identifica o aplicativo a ser baixado). O resultado é um arquivo APK, confirmando a correta execução do primeiro módulo. Testes foram realizados para explorar o potencial de aceleração do *download* através do uso do paralelismo.

Assim como o primeiro módulo, a funcionalidade do segundo já foi implementada, testada e validada. Este módulo utiliza a ferramenta Androguard para realizar uma engenharia reversa nos APKs para extrair todas as suas informações e características utilizadas em análises estáticas. O Androguard possui uma documentação[15] explicando todas as suas funcionalidades. Para iniciar a validação, o Androguard foi utilizado para realizar um teste de extração de características estáticas de um aplicativo Android. O código da ferramenta recebe como entrada um arquivo APK e, ao final, gera uma lista de características que podem ser armazenadas em um arquivo de texto, por exemplo.

O teste baseou-se em utilizar o Androguard para executar uma análise de código *byte* para extrair chamadas de API do aplicativo denominado Qubla, utilizando o método CG[16]. O método CG *(call graph)* tem a função de criar um grafo que representa todas

[15]https://Androguard.readthedocs.io/en/latest/
[16]https://androguard.readthedocs.io/en/latest/tools/androcg.html

as relações das chamadas de API presentes na aplicação. Essa abordagem permite uma análise abrangente do comportamento do aplicativo, uma vez que cada nó do grafo corresponde a uma chamada de API, garantindo a extração completa de todas as ocorrências dessa categoria. Ao final, foram extraídas 18895 chamadas de API da aplicação, onde foram armazenadas em um arquivo de texto. Para validar essa extração, foi comparado o número de características extraídas do código de extração com a do sistema Malscan[17] (i.e., outra ferramenta de extração de características que extrai chamadas de API), no qual resultou na mesma quantidade.

O teste realizado com o Androguard mostrou ser eficiente na extração de características estáticas de um aplicativo Android. Estas características podem ser utilizadas para a detecção de *malwares* ou para outras tarefas relacionadas a segurança em aplicativos Android. Os resultados do teste de extração estão disponíveis no repositório Github[18].

A partir da validação de todas as ferramentas e percepção que a sua integração permitiu a construção de um *dataset* eficiente que poderia ser utilizado para a detecção de *malwares* no Android, definiu-se que a arquitetura proposta com essas ferramentas foi viável.

4.3 Análise e Modelagem da Ferramenta

Seguindo as boas práticas de desenvolvimento de sistemas e da área de engenharia de *software*, nesta seção será apresentado a análise e modelagem para a construção do sistema da ferramenta ADBuilder. Desta forma seguiram-se os seguintes passos: a primeira etapa foi a definição do modelo de negócio baseada na arquitetura proposta e validada do sistema. A partir daí foram realizados o levantamento dos requisitos, sendo buscados requisitos funcionais, não funcionais e de sistemas, necessários para a construção da ferramenta. A partir dos requisitos, foi implementado o diagrama de casos de uso geral da ferramenta. Com os casos de uso, a próxima etapa foi implementar o diagrama de classes conceitual, e por fim, o diagrama de sequência.

[17] https://github.com/malscan/malscan

[18] https://github.com/Overycall/ADBuilder-TCC/tree/main/tests/Comparações%20de%20características/Outras%20comparações

4.3.1 Levantamento de Requisitos Funcionais, Não Funcionais e de Sistema

Conforme o (PRESSMAN; MAXIM, 2021), os requisitos são partes elementares para a construção de um sistema. Os requisitos dividem-se em três tipos: funcionais, não funcionais e de sistema. Os requisitos funcionais são aqueles que descrevem o que o sistema deve fazer ou realizar. Eles especificam as funcionalidades que o sistema deve ter, as entradas que ele deve aceitar, as ações que deve realizar e as saídas que deve produzir.

Os requisitos não funcionais são aqueles que descrevem as características do sistema que não são diretamente relacionadas à funcionalidade, mas que afetam a qualidade do sistema. Eles incluem requisitos como desempenho, escalabilidade, segurança, usabilidade, disponibilidade, entre outros.

Os requisitos de sistemas são aqueles que descrevem as características técnicas do sistema, como requisitos de *hardware*, *software*, rede, capacidade de armazenamento, compatibilidade de sistemas e outras características específicas do sistema.

A técnica utilizada para o levantamento de requisitos foi o *brainstorming*. O co-orientador e o grupo de projeto se reuniram em uma sala de reuniões para realizar uma sessão de *brainstorming* para construir a ideia inicial da ferramenta ADBuilder. O objetivo da sessão era gerar ideias para funcionalidades e características que a ferramenta deveria ter. Antes da sessão, o Co-orientador preparou uma lista de tópicos e um documento para registrar as ideias geradas.

A sessão começou com o Co-orientador definindo o tema da ferramenta ADBuilder como "Ferramenta de construção automatizada de *datasets* atuais para detecção de *malwares* no Android". Ele então pediu aos participantes que gerassem ideias para funcionalidades e características que a ferramenta deveria ter. As ideias foram escritas em um documento, durante a geração de ideias. Depois de gerar as ideias, o grupo as organizou por temas e classificou por importância. Eles discutiram as ideias geradas e determinaram quais deveriam ser incluídas no conjunto final de requisitos.

Vale ressaltar, que ao realizar o *brainstorming* para levantar os requisitos da ferramenta, é importante considerar critérios de validação e descartes de ideias para garantir que somente as melhores sejam selecionadas e utilizadas. Os critérios de validação incluem a relevância para a construção do *dataset*, a capacidade de integração com as etapas já definidas, a viabilidade técnica, a possibilidade de automação e a eficiência. Todas as ideias que atenderem a esses critérios devem ser consideradas e avaliadas com mais profundidade para serem implementadas. Já os critérios de descarte

incluem a irrelevância para a construção do *dataset*, a inviabilidade técnica, a falta de integração com as etapas já definidas e a impossibilidade de automação. Todas as ideias que não atenderem a esses critérios devem ser descartadas e não devem ser consideradas para a implementação. Guiando-se por esses critérios de validação e descarte, é possível garantir que a ferramenta ADBuilder seja desenvolvida de maneira eficiente e que ofereça as melhores soluções para a construção de *datasets* para detecção de *malwares* no Android. As Tabelas 5, 6 e 7, a seguir, apresentam a descrição dos requisitos funcionais, não funcionais e de sistemas da ferramenta ADBuilder, respectivamente.

Tabela 5 – Requisitos Funcionais [RF]

Código do Requisito	**Descrição**	**Prioridade**
[RF001] Coleta de APKs	O sistema deve coletar aplicativos Android normais e maliciosos através do repositório AndroZoo	Essencial
[RF002] Extração de características	O sistema deve extrair características estáticas dos APKs através da ferramenta Androguard	Essencial
[RF003] Rotulação de APKs	O sistema deve rotular os APKs como maliciosos ou não através do serviço do VirusTotal	Essencial
[RF004] Geração do *dataset*	O sistema deve gerar o *dataset* final concatenando os dados de extração e rotulação	Essencial
[RF005] Salvar estatísticas	O sistema deve salvar dados estatísticos da ferramenta como tempo de execução, gasto de RAM e CPU	Importante
[RF006] Armazenamento do VirusTotal	O sistema deve armazenar os dados das análises do VirusTotal em arquivos JSONs	Essencial
[RF007] Armazenamento em CSV	O sistema deve armazenar os dados das características extraídas e do *dataset* final em um arquivo CSV	Essencial
[RF008] Salvar chamadas de API	O sistema deve salvar a extração crua sem limpeza das chamadas de API em um arquivo de texto compactado	Desejável
[RF009] Interface amigável	O sistema deve ter uma interface fácil de usar para que os usuários possam configurar e iniciar o processo de construção do *dataset*.	Essencial

Fonte: Autor (2022).

Tabela 6 – Requisitos Não Funcionais [RNF]

Código do Requisito	**Descrição**	**Prioridade**
[RNF001] Desempenho	A ferramenta deve ser capaz de coletar, extrair, rotular e gerar o *dataset* de forma eficiente e rápida	Essencial
[RNF002] Escalabilidade	A ferramenta deve ser capaz de lidar com grandes volumes de dados, caso a necessidade surgir	Essencial
[RNF003] Compatibilidade	A ferramenta deve ser compatível com os sistemas operacionais e as ferramentas comuns usadas pelos usuários	Importante
[RNF004] Manutenibilidade	A ferramenta deve ser fácil de manter e atualizar para correção de bugs ou adição de novos recursos	Essencial
[RNF005] Documentação	A ferramenta deve incluir uma documentação detalhada, incluindo instruções de uso e explicações sobre as características extraídas e rotulações usadas	Essencial
[RNF006] Fiabilidade	A ferramenta deve ser capaz de fornecer resultados precisos e consistentes ao rotular os APKs como maliciosos ou não	Essencial
[RNF007] Usabilidade	A ferramenta deve ser fácil de se utilizar pelos usuários	Essencial
[RNF008] Linguagem de Programação	A ferramenta deve ser implementada em Python, Shell ou Java	Essencial

Fonte: Autor (2022).

Tabela 7 – Requisitos de Sistema [RS]

Código do Requisito	**Descrição**	**Prioridade**
[RS001] SO compatível	A ferramenta deve ser executada em sistemas operacionais Windows e Linux	Importante
[RS002] Espaço de Armazenamento	A ferramenta deve ter acesso ao espaço de armazenamento suficiente para armazenar os arquivos de dados gerados	Essencial
[RS003] Recursos de *Hardware*	A ferramenta deve ter acesso a recursos de *hardware* suficientes, como memória RAM e CPU, para processar os dados de maneira eficiente	Essencial
[RS004] Conexão com a Internet	A ferramenta precisa de conexão com a Internet para baixar os APKs e para utilizar o serviço do VirusTotal	Essencial

Fonte: Autor (2022).

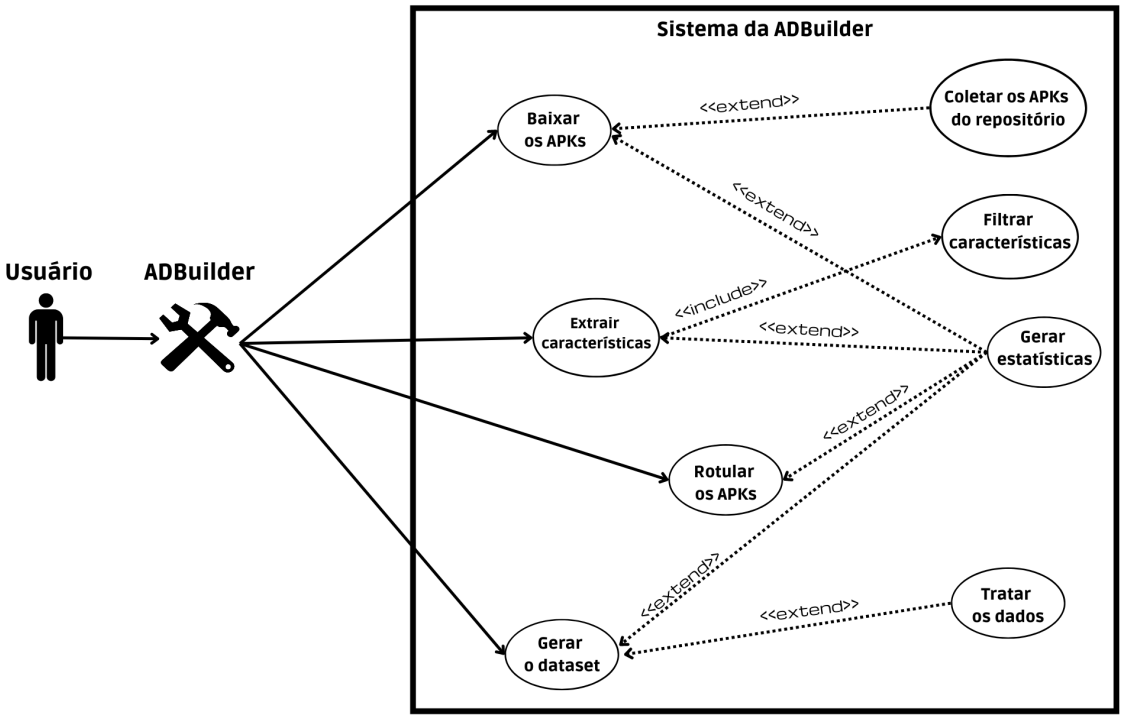
A partir do levantamento dos requisitos, foi possível identificar todas as necessidades e prioridades do sistema, tanto funcionais quanto não funcionais e de sistemas. Os requisitos funcionais forneceram informações sobre as funcionalidades que o sistema deve ter, enquanto os requisitos não funcionais forneceram informações sobre a qualidade do sistema, e os requisitos de sistemas forneceram informações sobre as características técnicas do sistema. A combinação desses requisitos permitiu uma compreensão completa do que o sistema deve fazer e como ele deve ser, permitindo que seja desenvolvido um sistema que atenda às necessidades dos usuários. Além disso, esses requisitos serão usados como base para o desenvolvimento do sistema, garantindo que as funcionalidades e características desejadas sejam implementadas. Com base nos requisitos, iniciou-se a implementação dos casos de uso, como será apresentado na seção 4.3.2.

4.3.2 Casos de Uso

O diagrama de caso de uso, segundo o autor (JACOBSON; NG, 2004), é uma forma de representar a ferramenta e os elementos que a compõe. Em outras palavras, um diagrama de caso de uso é uma técnica de modelagem de sistemas que permite representar graficamente as interações entre os atores e as funcionalidades do sistema. Ele é utilizado

para descrever os casos de uso, ou seja, as maneiras pelas quais os usuários utilizam o sistema. Um diagrama de caso de uso é composto por atores, casos de uso e relações entre eles. Os atores são as entidades externas ao sistema que interagem com ele, os casos de uso representam as funcionalidades do sistema, e as relações mostram como os atores interagem com os casos de uso. A Figura 6 descreve as funcionalidades da ferramenta ADBuilder, graficamente.

Figura 6 – Diagrama de Caso de Uso.



Fonte: Autor (2022)

É possível observar na Figura 6, que em um primeiro momento, foi preciso especificar os aplicativos que se deseja baixar para serem coletados a partir do repositório AndroZoo. Em seguida, para extrair as características foi preciso antes filtrar quais foram as mais relevantes para serem adicionadas ao *dataset*. Depois, foi necessário realizar a rotulação dos aplicativos, onde foi utilizado um serviço externo para isso. Por fim, para gerar o *dataset* foi necessário já passar pelas etapas de extração e rotulação, para que seus dados resultantes sejam concatenados e adicionados ao conjunto de dados final. Com o *dataset* gerado, foi preciso tratar seus dados (e.g., remover amostras duplicadas). Além

disso, foi possível observar no diagrama que foram gerados estatísticas da ferramenta após cada execução de módulo (i.e., baixar, extrair, rotular e gerar), sendo então fornecidos os dados sobre os tempos de execução e quantidade de memória RAM.

Os *datasets* gerados pela ferramenta ADBuilder poderão ser utilizados para alcançar diversos objetivos. Alguns podem incluir:

- **Treinamento de modelos de classificação:** Os pesquisadores e desenvolvedores podem usar o *dataset* gerado para treinar modelos de classificação de *malwares*. Eles podem usar técnicas de aprendizado de máquina para treinar esses modelos para detectar *malwares* com base nas características dos APKs.
- **Avaliação de técnicas de detecção de *malwares*:** os pesquisadores e desenvolvedores podem usar o *dataset* gerado para avaliar a eficácia de técnicas de detecção de *malwares* existentes e desenvolver novas técnicas.
- **Análise estatística:** os pesquisadores e desenvolvedores podem usar o *dataset* gerado para realizar análises estatísticas para entender melhor os tipos de *malwares* que estão presentes no ecossistema Android e como eles evoluem ao longo do tempo.
- **Comparação com outros conjuntos de dados:** os pesquisadores e desenvolvedores podem comparar os dados gerados com outros conjuntos de dados já existentes para comparar as características e as rotulações dos APKs e entender como os dados gerados se comparam a outras fontes.
- **Comunidade científica:** O *dataset* gerado pode ser compartilhado com a comunidade científica para ser utilizado em pesquisas e desenvolvimento de novas técnicas de detecção de *malwares* para dispositivos móveis.

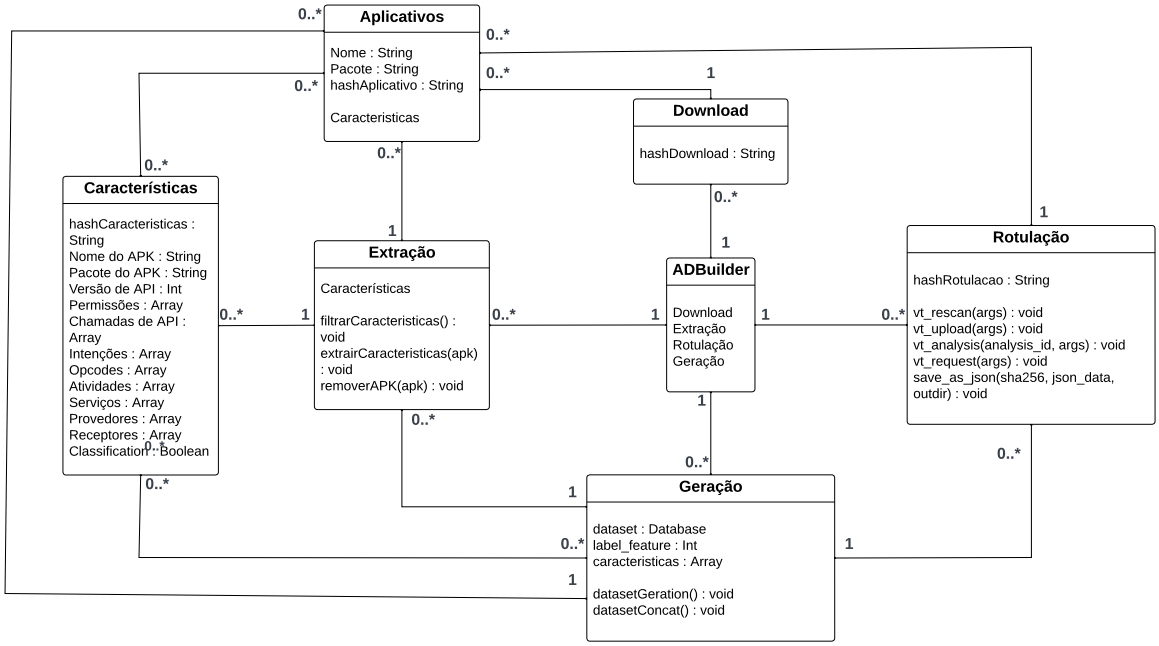
Esses são alguns dos possíveis usos da ferramenta, outros podem ser explorados e desenvolvidos de acordo com as necessidades específicas de cada caso. A análise do diagrama de caso de uso permitiu identificar todas as funcionalidades necessárias para o desenvolvimento da ferramenta de geração de *datasets* para detecção de *malwares* em dispositivos Android. Em seguida, foi elaborado o diagrama de classe conceitual.

4.3.3 Diagrama de Classe Conceitual

Segundo (FOWLER, 2004), um diagrama de classes conceitual é uma técnica de modelagem de sistemas que permite representar graficamente a estrutura de classes de um

sistema. Ele é usado para entender como os objetos de um sistema se relacionam entre si e como os dados e comportamentos são distribuídos entre as classes. O diagrama também ajuda a identificar o conceito das classes e atributos do sistema, bem como as relações existentes entre eles. A Figura 7, a seguir, apresenta o diagrama de classes conceitual da ferramenta ADBuilder.

Figura 7 – Diagrama de Classes Conceitual.



Fonte: Autor (2022)

O diagrama 7 apresenta a estruturação dos componentes da ADBuilder e a relação entre eles. Destaca-se que os módulos de *Download*, Extração, Rotulação e Geração são parte da ferramenta.

Quando um usuário realiza uma solicitação através da ferramenta ADBuilder, o sistema ativa os módulos de *Download*, Extração, Rotulação e Geração para atender à solicitação. Primeiro, o módulo de *Download* é acionado, permitindo ao usuário baixar múltiplos aplicativos Android. Em seguida, o módulo de Extração é acionado para recuperar diversas características de cada aplicativo baixado. As características podem ser diferentes entre os aplicativos, e cada extração pode resultar em várias características. O módulo de Rotulação é utilizado para classificar os aplicativos, e pode ser acionado diversas vezes para rotular vários APKs. O módulo de Geração, por sua vez, manipula os dados de rotulação e os dados de extração para construir o *dataset*. O módulo de Geração pode ser executado várias vezes, permitindo a geração de diversos *datasets* com

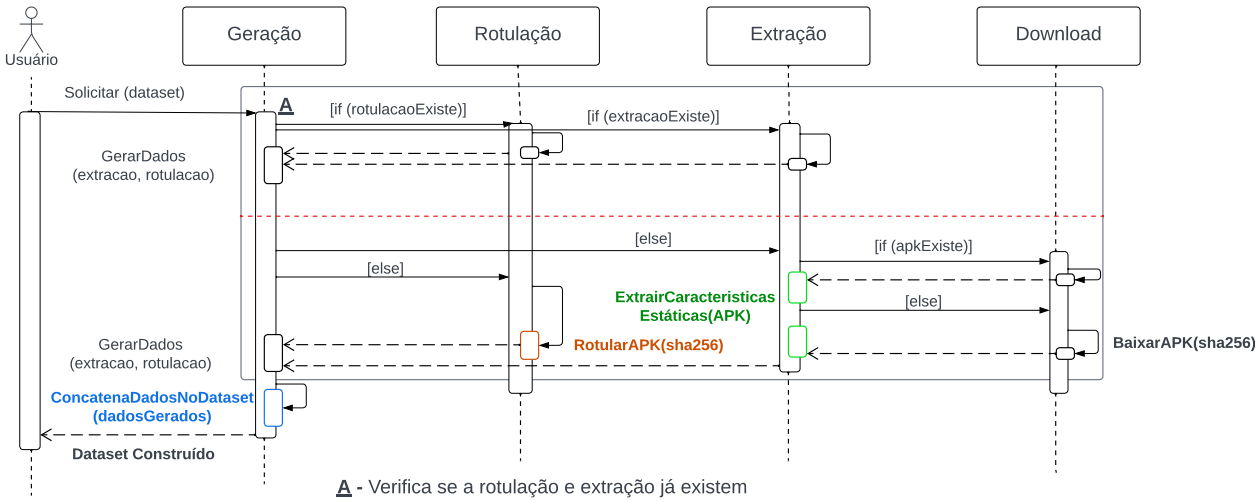
diferentes características e aplicativos. Assim, a ADBuilder trabalha com todos os seus módulos para atender à solicitação do usuário, possibilitando a geração de *datasets* para análise de *malwares* Android com base nas características extraídas dos aplicativos e nas rotulações realizadas pelo módulo de Rotulação.

A próxima etapa após a construção do diagrama de classes conceitual é elaborar o diagrama de sequência da ferramenta. Isso será abordado na seção 4.3.4 a seguir.

4.3.4 Diagrama de Sequência

Os diagramas de sequência são uma técnica de modelagem de sistemas que permitem representar de forma visual as interações entre os objetos de um sistema, de acordo com (DELLIGATTI, 2013). Eles são usados para descrever as interações entre os objetos ao longo do tempo, mostrando como eles se comunicam entre si, quais mensagens eles trocam e em que ordem isso acontece. Um diagrama de sequência é composto por objetos, mensagens e linhas de tempo. A Figura 8, a seguir, apresenta o diagrama de sequência da ferramenta ADBuilder.

Figura 8 – Diagrama de Sequência.



Fonte: Autor (2022)

Na Figura 8, é possível observar que existem 4 (quatro) módulos: geração, rotulação, extração e *download*. Quando surge uma necessidade do usuário por um *dataset*, ele configura e executa a ferramenta que comunica ao módulo de geração que construa um conjunto de dados com os aplicativos Android referenciados pela lista de

sha256 fornecida pelo usuário. Para construir o *dataset*, o módulo de geração precisa dos dados da rotulação e da extração para um mesmo APK. Então é realizada uma verificação para verificar se a rotulação e a extração já processaram esses dados. Se os dois dados existem, então o módulo de geração consegue manipular e tratá-los, concatenando-os para adicioná-los ao *dataset* já em construção.

Se os dados de extração e rotulação ainda não existirem, será preciso que cada módulo execute as funções de geração. Para isso, através do SHA256 fornecido, o módulo de rotulação consegue rotular o respectivo APK, sinalizando para o módulo de geração que seu processo acabou. Já o módulo de extração, precisa verificar se o arquivo APK existe na fila de *download*, se existir, extrai as características e envia esses dados para o módulo de geração. Se o APK não existir, então o módulo de *download* baixa o aplicativo e sinaliza para a extração que o arquivo está pronto para ser processado. Em seguida, o módulo de extração extrai as características do APK e envia os dados para o módulo de geração. Com isso, o módulo de geração consegue processar os dados e construir o *dataset*.

4.3.5 Armazenamento dos Dados

A ADBuilder utiliza um sistema de filas baseado no princípio de produtor e consumidor para garantir a eficácia e eficiência na construção dos *datasets*. Cada módulo será responsável por produzir seus dados, armazenando-os em sua própria fila, passando os dados para o módulo seguinte, que os consome. Os dados que foram gerados pelo primeiro módulo (*download* de APKs) são arquivos .apk que foram armazenados na fila de *download*.

O segundo módulo (extração de características) consome os aplicativos baixados e produz novos dados. Inicialmente, a abordagem adotada para armazenar as características extraídas era utilizar arquivos no formato CSV. Essa estratégia permitia manter a organização dos dados e facilitava o acesso às informações relevantes. O formato CSV também oferecia compatibilidade com a maioria dos programas de análise de dados, tornando a leitura e a manipulação dos dados mais fáceis. No entanto, à medida que o projeto avançou, percebeu-se que armazenar as características em arquivos JSON acelerava significativamente a leitura e a manipulação dos dados. O JSON possui vantagens como estrutura hierárquica, suporte a tipos de dados complexos e capacidade de representar dados de forma mais compacta. Essa mudança para o formato JSON

possibilitou realizar filtragens e estudos mais aprofundados com maior facilidade e rapidez. Além disso, todas as chamadas de API foram extraídas e armazenadas em um arquivo de texto compactado (.zip). No entanto, no *dataset* final, serão incluídas apenas as chamadas de API que estão presentes no pacote oficial do Android e do Java. Essa filtragem se tornou necessária devido ao potencial alto volume de ocorrências das chamadas de API, o que poderia tornar o processo de construção do *dataset* mais demorado e oneroso. A decisão de armazenar as chamadas de API em um arquivo separado permite realizar análises mais detalhadas e estudos mais aprofundados.

O terceiro módulo (rotulação) recebe a lista de SHA256 que identifica os APKs e gera um arquivo JSON contendo os dados resultantes da análise do VirusTotal para cada APK, armazenando-os em sua própria fila. Por fim, o quarto módulo (geração de *dataset*) consome os dados gerados pelos módulos de extração e rotulação e gera o *dataset* final como um arquivo CSV, armazenando-o em uma pasta dentro de sua fila de geração.

Esse mecanismo de filas garante que os dados sejam processados de forma ordenada e eficiente, sem que haja necessidade de esperar que todos os dados sejam gerados antes de processá-los. Ademais, evita o problema de travar o processamento devido ao acúmulo de dados não processados, o que garante o bom desempenho da ferramenta. Essa estratégia de armazenamento ainda possibilita a reutilização desses dados para novos estudos sem a necessidade de processá-los novamente, o que garante a eficiência e economia de tempo.

4.4 Estudo das tecnologias

Em relação as tecnologias que foram utilizadas na implementação do projeto, foi realizado uma investigação tendo como objetivo encontrar as melhores opções, considerando a viabilidade e finalidade da ferramenta. De acordo com a Tabela 8, é possível observar as tecnologias candidatas a serem adotadas neste projeto. Para isso foi necessário realizar uma investigação minuciosa para selecionar as tecnologias com melhor aderência.

Tabela 8 – Tecnologias que poderão ser adotadas.

	Tecnologias que poderão ser adotadas
Versionamento do Projeto	*GitHub* ou *GitLab*
Desenho da Arquitetura	*Lucidchart*, *App Diagrams* e/ou *Figma*
Linguagem de Programação	*Java*, *Python* e/ou *Shell Script*
Ferramentas para Extração de Características	*AndroGuard*, *APKFile*, *Apktool*, *Appmon* ou *AndroPyTool*
Serviços para Rotulação dos Aplicativos	*VirusTotal*, *Avira Antivírus* ou *SandDroid*
Repositórios	AndroZoo, Koodous ou Google Play

Fonte: Autor (2022)

De acordo com a Tabela 8, primeiramente foi necessário escolher entre as ferramentas *Github*[19] ou *Gitlab*[20] para versionar o projeto. Segundo, foi importante selecionar uma ferramenta para desenhar os diagramas de arquitetura e implementação da ferramenta, para auxiliar no entendimento do funcionamento do processo de construção do *dataset*, como o Figma[21], Lucidchart[22] ou AppDiagrams[23]. Em seguida, foi necessário selecionar uma linguagem de programação com melhor aderência para implementar a ferramenta ADBuilder. Depois, veio a seleção das ferramentas de extração de características estáticas e de rotulação dos aplicativos Android, as quais foram implementadas e incorporadas nos módulos de extração e rotulação, respectivamente. Por fim, foi escolhido um repositório adequado para realizar o *download* dos APKs.

A seleção de um repositório de APKs é muito importante pois é ele quem vai definir a qualidade das amostras que serão inclusas no *dataset*. As opções levantadas foram o AndroZoo, Koodous e Google Play Store. *Java*[24] é a linguagem padrão de escrita de aplicativos para Android, ao qual fornece suporte para realizar a engenharia reversa dos mesmos, extraindo informações do arquivo do Android manifesto. Já a linguagem *Python*[25], além de conter um arsenal de bibliotecas robustas que podem ser integradas no projeto, possui uma comunidade ativa e crescente que disponibiliza muitos recursos e soluções para problemas que ocorrem durante o trabalho. Para extração de

[19]https://github.com
[20]https://gitlab.com
[21]https://www.figma.com
[22]https://lucid.app
[23]https://app.diagrams.net
[24]https://docs.oracle.com/en/java/
[25]https://docs.python.org/pt-br/3/tutorial/

características existem as ferramentas *Androguard*[26], *AndroPyTool*[27] e *Appmon*[28] para serem implementadas em *Python*, e há as ferramentas *APKFile*[29] e *Apktool*[30] que podem ser utilizadas na linguagem de programação *Java*. Dentre as ferramentas e serviços de rotulação, há: o serviço online do *VirusTotal*, que disponibiliza uma API para ser utilizada na ferramenta, trazendo informações da análise dos aplicativos através de mais de 60 *scanners*; o antivírus do Avira, que pode ser utilizado para identificar se um aplicativo é malicioso; e o *SandDroid* que disponibiliza uma análise tanto estática quanto dinâmica dos aplicativos.

4.4.1 Repositórios de APKs

Visto a finalidade deste trabalho, a construção de um *dataset* para detecção de *malwares* Android, é necessário realizar o *download* dos arquivos binários dos aplicativos (.apk). Esses arquivos podem ser encontrados em repositórios de aplicativos Android como a Google Play Store[31], o Koodous[32] e o AndroZoo[33]. Koodous e AndroZoo representam alternativas à Google Play Store. Por exemplo, o AndroZoo cataloga e analisa milhões de APKs desde 2011 (ALLIX et al., 2016). Muitos APKs que não estão mais disponíveis na Google Play Store estão disponíveis no AndroZoo, e isso pode acontecer devido a diversos motivos, como evolução das versões (e.g., uma versão mais recente substitui a antiga no repositório), ou até mesmo pelo aplicativo ter sido detectado com um comportamento malicioso. Portanto, repositórios como o AndroZoo podem representar um acervo rico para a seleção e o *download* de amostras de aplicativos. Por fim, repositórios como o AndroZoo oferecem mais recursos e menos limitações para o *download* de APKs, conforme a Tabela 9, a qual foi implementada com o intuito de coletar dados de diferentes repositórios e compará-los para selecionar o mais adequado.

[26] androguard.readthedocs.io
[27] https://github.com/alexMyG/AndroPyTool
[28] https://github.com/dpnishant/appmon
[29] https://github.com/CalebFenton/apkfile
[30] https://ibotpeaches.github.io/Apktool/
[31] <https://play.google.com/store>
[32] <https://koodous.com/apks>
[33] <https://androzoo.uni.lu>

Tabela 9 – Repositórios.

Repositório/URL	**Quantidade de APKs**	**Limitações**	**Metadados**	**Disponível**
Google Play Store / play.google.com/store	2.900.000+	1- Limite de 15 *downloads* diários. 2- Necessidade de utilizar um dispositivo ou emulador para baixar o APK	Versão do app, Versão do Android, Data de lançamento, Data de atualização, Número de *downloads*, Avaliações dos usuários, Faixa etária, Categorias	Sim
Koodous / koodous.com/apks	-	1- No serviço gratuito, há um limite de 5 requisições diárias, para analisar um APK. 2- Não é possível baixar os APKs pelo site. 3- A rotulagem dos aplicativos é realizada pelos próprios usuários.	SHA256, SHA1, MD5, Tamanho do app, Nome do pacote, Versão do código, Data de última visualização, URLs, Intenções, Serviços, Receptores, Provedores, Bibliotecas, Atividades, Certificados, Permissões, API Calls, Versões de API.	Sim
AndroZoo / androzoo.uni.lu	21.736.680 (16/01/2023)	1- Permite utilizar 20 *downloads* em paralelo com uma API Key (40 se você estiver fora da Europa)	SHA256, SHA1, MD5, Data do arquivo DEX, Tamanho do app, Nome do pacote, Versão do código, Detecções do VirusTotal, Data da detecção, amanho do arquivo DEX, Origem da fonte de dados	Sim
Anzhi / anzhi.com	400.000+	1- Idioma chinês por padrão. 2- Plataforma para software Android 4.0 ou superior.	Versão do android, Tamanho do app, Data da atualização, Autor, Número de *downloads*, Avaliação dos usuários	Sim
AppChina / appchina.com	-	1- Idioma chinês por padrão. 2- Não é atualizado com frequência	Tamanho do app, Número de *downloads*, Avaliação dos usuários, Histórico de versões, Última atualização, Formato, Categoria, Requisitos, Desenvolvedor	Sim
1mobile / market.1mobile.com	800.000+	1- Requer no mínimo um sistema operacional: Android 2.3	-	Não
FreewareLovers / freewarelovers.com	1.711	-	Desenvolvedor, Categorias, Última versão do app, Quantidade de versões, Data de lançamento, Data de atualização	Sim
HiAPK / hiapk.com	-	-	-	Não
F-Droid / f-droid.org	4.598	-	Versão do app, Data de lançamento, Versão do Android, Permissões, Autores, Licenças	Sim

Fonte: Autor (2022)

Atualmente, lojas oficiais como a Google Play Store impõem diversas limitações para o *download* de aplicativos, como a necessidade de utilizar um dispositivo ou emulador como o Android Studio, a falta de APIs para automação (e.g., *download* simplificado de APKs), limites diários (que podem ser modificados via pagamento) e a não disponibilização de conjuntos de metadados mais expressivos (e.g., informações

para rotulação dos APKs). Repositórios como o Koodous também possuem restrições. Na versão gratuita, não é possível realizar o *download* ou as análises dos APKs, esse privilégio só é possível para usuários assinantes da API paga. Além disso, as informações de rotulação dos aplicativos não são confiáveis, pois são realizadas manualmente pelos próprios usuários do repositório. O usuário também precisa estar logado para ver as informações dos APKs.

Após selecionar o repositório, é importante realizar a coleta dos metadados dos aplicativos para selecioná-los seguindo alguns critérios (e.g., atualidade e rotulação das amostras). Uma informação importante, tipicamente contida nos metadados dos aplicativos, é o resumo criptográfico (e.g., SHA256), que permite identificar unicamente cada aplicativo. Apesar de repositórios como a Google Play não informarem explicitamente o resumo criptográfico, ele pode ser gerado após o *download* do APK. Há trabalhos de construção de *datasets*, como (CATAK; YAZI, 2019), (DÜZGÜN et al., 2021) e (BORGES et al., 2021), os quais utilizam apenas o MD5 para identificar as amostras, o que pode causar problemas de colisão, como é bem conhecido na literatura, tomando como referência os trabalhos (KLIMA, 2005) e (KLIMA, 2006).

Observa-se ainda na Tabela 9 que o repositório AndroZoo é o único que disponibiliza, em seus metadados, a informação da origem de onde foi coletada a amostra, além de apresentar a quantidade de detecções de *scanners* do VirusTotal, assim como a data em que foi realizada essa detecção, o que auxilia na etapa da rotulação dos *malwares* e, também, na validação da atualização e confiabilidade dos dados. Dessa forma, o AndroZoo foi o repositório candidato mais forte para ser integrado na ferramenta, pois além de fornecer uma diversidade de metadados, fornece também uma API que pode ser utilizada para simplificar e automatizar o *download* dos APKs, com uma limitação diária bem baixa em relação ao paralelismo e sem limitações em relação a quantidade de aplicativos que podem ser baixados.

O repositório AndroZoo, nos dias de hoje, disponibiliza mais de 22 milhões de aplicativos Android para *download*. Além disso, disponibiliza um conjunto de metadados que apresenta os seguintes dados:

- **SHA256, SHA1 e MD5** - Resumos criptográficos que identificam unicamente um arquivo APK;
- **Dex Date** - Data que é anexada ao arquivo DEX do APK. Se não existir uma data, por padrão é definida como 1980;
- **Pacote** - Nome do pacote do respectivo aplicativo (e.g., com.facebook.katana);

- **Código de versão** - Código da versão do aplicativo;
- **Detecções do VT** - Esse dado é um número inteiro que representa a quantidade de *scanners* do VirusTotal que identificaram o APK como malicioso;
- **Data da detecção do VT** - Data que indica quando foi realizada a última análise de detecção do respectivo APK no VirusTotal;
- **Tamanho DEX** - Tamanho do arquivo DEX do APK, em bytes;
- **Mercados** - Origem da fonte de dados, que informa de onde o respectivo APK foi retirado.

4.4.2 Serviços para Rotulação de Aplicativos

Diversos trabalhos utilizam amostras de *malwares* de outros *datasets*, mais antigos, ou metadados de rotulação como os contidos no AndroZoo. O problema é que ambos os casos representam dados defasados para a detecção de *malwares*. Por exemplo, os dados de rotulação do AndroZoo podem estar significativamente desatualizados. Na prática, um aplicativo que era considerado benigno em 2021 (0 detecções) pode ser considerado como maligno em 2022 (e.g., 20 detecções). Portanto, é imprescindível a atualização dos dados de rotulagem na construção e utilização de um *dataset* para detecção de *malwares*. Sem essa atualização, muito provavelmente o modelo será otimizado para dados incorretos, levando a vieses e falhas grandes de detecção.

Existem diversas ferramentas *desktop* (e.g., Microsoft Windows Defender, Avira, Norton, Avast e Dr Web) e também serviços online (e.g., VirusTotal, Koodous, SandDroid) que podem ser utilizados para a etapa de rotulagem. Atualmente, o VirusTotal[34] é um dos serviços mais completos e mais frequentemente utilizados para a rotulagem de *malwares*. Através de uma API simples, qualquer usuário pode submeter aplicativos para análise. O serviço conta com um arcabouço de mais de 60 *scanners*, sendo que cada um deles irá analisar o aplicativo. O relatório final do VirusTotal, em formato JSON, irá identificar o resultado de cada um dos *scanners*. Apesar disso, é importante destacar que alguns trabalhos recentes (e.g., (DHALARIA et al., 2021)) utilizam apenas ferramentas como o Avira, que podem ser limitadas em termos de quantidade de *scanners* disponíveis.

É possível utilizar o VirusTotal pelo site ou através de sua API, onde é necessário

[34] virustotal.com

utilizar uma chave obtida ao se cadastrar. No site é possível enviar um arquivo (e.g., .exe ou .apk), URL, endereço de IP ou a função *hash* que identifique o arquivo (e.g., SHA256), para ser analisado pelo sistema. O VirusTotal apenas consegue analisar o arquivo pelo seu *hash* se o mesmo já estiver contido em seu banco de dados, se não, é necessário fazer o envio do arquivo. Por exemplo, ao submeter um APK para análise, o VirusTotal retorna informações abrangentes sobre o aplicativo, incluindo detalhes como funções de *hash*, nome do pacote e tamanho do arquivo. Além disso, são fornecidos *insights* sobre seu comportamento, como as permissões solicitadas, intenções e atividades. O VirusTotal disponibiliza detecções, permitindo que o usuário identifique quais dos 62 *scanners* antivírus rotularam o aplicativo como malicioso. Porém, o resultado da análise retornada é a da última realizada, então é muito importante sempre analisar a data da última análise e solicitar uma reanálise, para que o sistema processe o arquivo novamente para retornar os dados de rotulação atualizados.

Além de realizar essa análise pelo site, enviando manualmente um arquivo de cada vez, também é possível utilizar a API do VirusTotal e automatizar as análises, enviando apenas o SHA256 do APK. Há um limite de 500 requisições diárias por chave de API. Esse limite está ligado também ao IP da máquina, então em um mesmo computador não será possível utilizar duas chaves de API diferentes para ultrapassar essa limitação. Em geral, o serviço do VirusTotal detalha muitos aspectos importantes que ajudam a identificar um *malware* e seus comportamentos. Porém, é importante manter um nível de confiança relativo sobre este serviço, como descrito em (WANG et al., 2019a). As detecções dos *scanners* estão em constantes alterações, seja pela evolução do aprendizado de máquina de cada antivírus ou da evolução dos *malwares* (SAHAY, 2020). Por exemplo, nos metadados do AndroZoo, a data das detecções do VirusTotal para os APKs em questão são antigas (e.g., 2018), sendo que muitos possuem detecções diferentes atualmente. Então, é preciso estar sempre atualizando as informações de rotulagem de um aplicativo para não incluir informações erradas em um *dataset* e, possivelmente, fornecer um aplicativo benigno para o treinamento de um modelo, sendo que agora é considerado maligno no VirusTotal, o que pode confundir o discernimento do modelo de aprendizado de máquina, por exemplo.

Assim como o VirusTotal, o Koodous pode ser utilizado para rotular *malwares*, porém é menos confiável, pois as rotulações são realizadas por votações entre os usuários cadastrados em seu site. Então, qualquer usuário pode rotular um aplicativo entre maligno ou benigno, o que pode levantar suspeitas em relação a veracidade dos dados. Além da

rotulação, o Koodous também retorna uma análise para um APK, contendo informações sobre as características (e.g., permissões, chamadas de API, intenções). É possível utilizar a API do Koodous através de uma chave obtida após o cadastro em seu site, retornando de sua análise um arquivo JSON, assim como o VirusTotal. Porém há diversas limitações que dificultam análises em grandes volumes, além de não permitirem mais o *download* de APKs e uso da API gratuitamente.

O SandDroid[35] foi outra opção de serviço. Nele é possível enviar o aplicativo Android, se esse ainda não estiver contido em seu banco de dados, para realizar uma análise estática e dinâmica, retornando o risco desse aplicativo ser um *malware*, suas características estáticas (e.g., permissões, componentes do aplicativo, chamadas de API sensíveis) e dinâmicas (e.g., tráfego de rede, informações de IP, SMS, monitoramento de chamadas de telefone). Esse serviço proporciona uma visão global do aplicativo malicioso, que além de retornar os dados da análise estática e dinâmica, informa dados dos modelos de aprendizado de máquina que classificaram o APK como malicioso e as permissões mais utilizadas por essas aplicações, assim como um top 20 das famílias dos *malwares*. Limitam-se em utilizar apenas o MD5 para identificar o aplicativo.

Por fim, conclui-se que o VirusTotal foi o serviço mais adequado para ser integrado na ferramenta ADBuilder para rotular os aplicativos, pois além de ser um dos mais utilizados pelas pesquisas levantadas, como (LI et al., 2017), (LASHKARI et al., 2018), (WANG et al., 2019b) e (DÜZGÜN et al., 2021), disponibiliza mais de 60 *scanners* para verificar os aplicativos e trazer uma análise atualizada e detalhada sobre. Além disso, o VirusTotal disponibiliza uma API completa que foi integrada na ADBuilder. Essa API possui uma documentação[36] detalhada sobre seu funcionamento, e isso facilitou o uso de suas funcionalidades.

4.4.3 Ferramentas para Extração de Características

Uma das etapas da construção de um *dataset* é extrair características dos aplicativos que poderão ser utilizadas posteriormente para os classificar entre maligno ou benigno, por exemplo. Enquanto a maioria dos *datasets* disponíveis contém uma lista limitada de características e categorias de características (e.g., apenas permissões), a relação de características e categorias pode ser extensa. Em termos de características

[35] http://sanddroid.xjtu.edu.cn
[36] https://developers.virustotal.com/reference/overview

estáticas, por exemplo, podemos ter categorias como permissões, intenções, atividades, provedores, receptores, serviços, *opcodes*, chamadas de API e outras *strings* (e.g., URLs) (PAN et al., 2020).

Existem diversas ferramentas que realizam a extração das características de um aplicativo Android. Por exemplo, ferramentas como Androguard[37] e APKtool[38] permitem a extração de características estáticas dos aplicativos. Estudos recentes apontam que a ferramenta Androguard é a mais completa para a extração de características estáticas (PONTES et al., 2021) e uma das mais utilizadas (PAN et al., 2020).

Existem também ferramentas como AndroPyTool (MARTÍN; LARA-CABRERA; CAMACHO, 2018), DroidBox (SUGUNAN; KUMAR; DHANYA, 2018) e SandDroid (TEAM et al., 2014), que permitem a extração de características dinâmicas (e.g., eventos do sistema, chamadas de API em tempo de execução, chamadas de sistema, tráfego de rede). No caso da ferramenta AndroPyTool, a execução é via linha de comando e tem como entrada um diretório contendo arquivos com extensão ".apk"(*Android Application Pack*), que é um arquivo de instalação de aplicativo destinado ao sistema operacional móvel Android, e tem como saída características dinâmicas estruturadas no formato CSV (i.e., arquivo separado por vírgula) e JSON. Já o DroidBox fornece uma *sandbox* móvel para analisar aplicativos Android em tempo de utilização, ou seja, enquanto o usuário utiliza o aplicativo, o sistema registra eventos de acesso ao sistema de arquivos, rede, interação com o sistema operacional e interação com outros aplicativos. O SandDroid[39] é um sistema WEB que realiza uma análise estática e dinâmica de um aplicativo, onde o usuário faz o *upload* de um arquivo APK e o sistema apresenta como saída um relatório *HTML* contendo uma lista de características estáticas e dinâmicas. O processo de extração de características pode considerar tanto características estáticas quanto dinâmicas. Idealmente, quanto mais completo o processo, melhor, pois haverão mais dados para a identificação de comportamentos anômalos ou padrões.

Observa-se que diferentes métodos de uma mesma ferramenta podem levar a resultados distintos. Por exemplo, o Androguard disponibiliza diferentes métodos para extrair características. Alguns trabalhos (e.g., (BAUMGÄRTNER et al., 2015) e (YU; AGARWAL; HONG, 2021)) utilizam os métodos *XREF_TO* e *XREF_FROM* do Androguard, que derivam do *dx.get_methods*. O método *dx.get_methods* é a saída mais crua dos dados das chamadas de API. Portanto, faz-se possível retornar cada método,

[37] <https://github.com/Androguard/Androguard>
[38] <https://ibotpeaches.github.io/Apktool>
[39] http://sanddroid.xjtu.edu.cn/

especificando qual objeto está chamando qual e vice-versa, e quais são os métodos que estão entre as chamadas de entrada e saída. Ademais, apresenta informações como *flags* de acesso (e.g., pública, privada, estática, construtor) e parâmetros dos métodos. Os métodos *XREFs* podem ser utilizados para filtrar as chamadas, seja as que são invocadas ou as que invocam. *TO* referencia o objeto que está chamando outro objeto. *FROM* referencia o objeto que está sendo chamado por outro objeto. A documentação do Androguard mostra exemplos mais detalhados[40]. Entretanto, o Androguard disponibiliza o método *CG*, que constrói um grafo de todas as chamadas de API do aplicativo. Na prática, ele irá identificar também todas as chamadas intermediárias, isto é, entre quaisquer dois pares de chamadas *XREF_TO* e *XREF_FROM*. O grafo permitirá, eventualmente, identificar certos comportamentos de aplicativos benignos e malignos, como a recorrência de certos fluxos de chamadas de API. Na prática, o método *CG* irá retornar uma lista mais completa e detalhada de todas as chamadas de API utilizadas no aplicativo. Tomando como exemplo o aplicativo *Pulse App Heart Rate Monitor*, enquanto os métodos *XREF_TO* e *XREF_FROM* irão retornar uma lista de 25806 chamadas de API, o método *CG* irá construir uma lista de 40105 chamadas de API. Portanto, o método *CG* foi selecionado para ser utilizado na ADBuilder.

Dentre todas as ferramentas de extração levantadas, o Androguard foi considerado a melhor opção para ser integrada na ADBuilder. O Androguard é escrito em Python, o que significa que é fácil de usar e de integrar com outras ferramentas. Além disso, o Androguard fornece uma ampla gama de funcionalidades para extrair características de APKs, incluindo informações sobre a estrutura do pacote, informações sobre os componentes do aplicativo, como classes e métodos, e informações sobre as permissões e chamadas de API. Isso é útil para projetos que precisam de um grande volume de características para treinar modelos de aprendizado de máquina. Por esses e outros motivos, o Androguard foi a opção mais adequada para ser integrada no módulo de extração da ferramenta ADBuilder.

A partir do mapeamento das tecnologias disponíveis e da seleção das ferramentas com melhor aderência para serem integradas neste projeto, foi possível definir as dependências e requisitos de execução. Essas informações serão abordadas em detalhes na seção 4.5.

[40]https://readthedocs.org/projects/Androguard/downloads/pdf/latest/

4.5 Requisitos de execução

A Tabela 10 apresenta um resumo das tecnologias, ferramentas e módulos utilizados na implementação. As linguagens Python (mantendo compatibilidade de código com as versões 3.8 e 3.9) e Bash foram utilizadas para a implementação dos módulos e das filas entre estes. A ferramenta e biblioteca AndroGuard foi utilizada na implementação do módulo de extração. A biblioteca *Networkx* é utilizada em conjunto com a AndroGuard para criar o grafo das chamadas de API. Para manipular os arquivos CSV, foi utilizada a biblioteca Pandas do Python. Em relação a rotulação, foi necessário utilizar a biblioteca *Requests* para realizar requisições GET. Por fim, para manipular os dados de execução e exibi-los adequadamente ao usuário, foi utilizado as bibliotecas *Termcolor* e *Pyfiglet*.

Tabela 10 – Requisitos de Linguagens, ferramentas e módulos.

	Nome	**Versão**
Linguagens	Python	3.8 ou 3.9
	Bash	5.0.17(1)-release
Ferramentas	Androguard	3.3.5
	Curl	7.68.0
Módulos Python	Lxml	4.5.0
	Pandas	1.3.5
	Networkx	2.2
	Termcolor	1.1.0
	Pyfiglet	0.8.post1
	Requests	2.22.0
	Numpy	1.22.3
Sistema Operacional	Ubuntu	20.04 LTS

Fonte: Autor (2023)

Foram planejados de forma organizada um conjunto de ambientes para os testes da ferramenta ADBuilder, utilizando diferentes distribuições, como o GNU/Linux Ubuntu 20.04 e 22.04. Os detalhes completos dos ambientes estão disponíveis no repositório do GitHub[41].

[41] https://github.com/Overycall/ADBuilder-TCC

4.6 Implementação

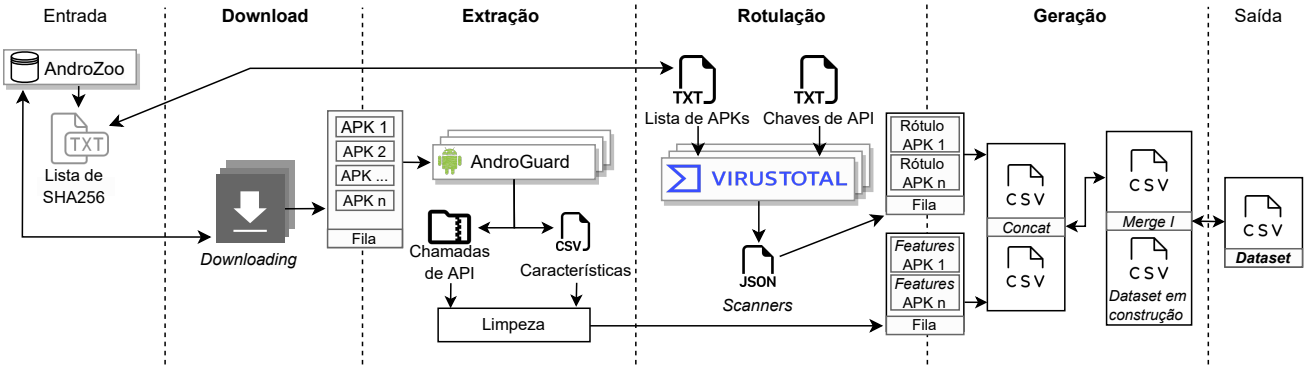
A ferramenta proposta foi composta por 4 (quatro) módulos independentes entre si: (a) o módulo de *download* (i.e., que realiza o *download* dos APKs); (b) o módulo de extração (i.e., que extrai as características estáticas dos APKs); (c) o módulo de rotulação (i.e., que rotula os APKs entre maliciosos ou não); e (d) o módulo de geração (i.e., que gera o *dataset* de fato). A implementação da ADBuilder foi realizada utilizando a linguagem de programação Python[42], por disponibilizar muitas bibliotecas e APIs que auxiliarão no processo de construção do *dataset*, e por ser mais simples de se integrar com outras ferramentas. Além do Python, foi utilizado a linguagem *Shell* para manipular os módulos e integrá-los.

Todos os módulos da ferramenta são executados simultaneamente, conversando entre si através de filas. Os módulos de rotulação e geração têm seus processos sendo executados sequencialmente. A geração necessita ser sequencial para construir o *dataset*, pois precisa dos dados tanto de rotulação quanto de extração de um mesmo APK para adicionar ao conjunto de dados final. Já o módulo de rotulação precisa se adaptar as limitações da API gratuita do VirusTotal, sendo preciso respeitar um limite de 500 requisições por dia e 4 requisições por minuto. Analisar um aplicativo equivale a uma requisição.

Os módulos de *download* e extração podem executar suas operações com paralelismo para acelerar o processo de construção do *dataset*. O paralelismo pode ser configurado antes de executar a ferramenta, via terminal. A Figura 9 a seguir apresenta o diagrama de implementação da ferramenta ADBuilder.

[42]https://www.python.org

Figura 9 – Implementação da ADBuilder.



Fonte: Autor (2023)

Observando a Figura 9 da implementação, o fluxo da ADBuilder inicia-se em receber uma lista de SHA256 dos aplicativos que se deseja adicionar ao *dataset*. Em seguida, eles são coletados do repositório AndroZoo e baixados paralelamente (número do paralelismo a ser definido pelo usuário), onde o módulo de *download* sinaliza para o componente de extração que seu processamento já terminou.

O módulo de extração espera a cada 5 segundos para verificar na fila de *download* se já possui algum arquivo APK pronto para ser processado por ele. Quando o arquivo estiver pronto, a extração inicia, sendo possível ser executada em paralelo (número de paralelismo a ser definido pelo usuário). Quando a extração para um APK terminar, é gerado um arquivo JSON contendo as características extraídas e um arquivo de texto compactado (.zip) contendo a extração crua das chamadas de API, que podem ser utilizadas em experimentos ou estudos posteriores.

As chamadas de API são filtradas antes de serem armazenadas no arquivo JSON, pois são a categoria de características que possui mais ocorrências em aplicativos Android. Em casos extremos, um aplicativo Android pode conter mais de 300.000 (trezentas mil) chamadas de API, o que representa um grande volume de dados e que pode elevar a necessidade de utilizar um *hardware* com maior capacidade de processamento. Além disso, dentre todas as chamadas de API extraídas, existem muitas que não fazem parte do pacote Android ou Java, e que representam métodos normais da linguagem de programação (e.g., *clone*, *run*, *wait*). Com base nestas observações, optou-se realizar um filtro na extração. Quando o módulo de extração terminar seu processamento ela sinaliza para o módulo de geração que os dados das características estão prontos.

O módulo de rotulação, assim como o *download*, recebe a lista de SHA256 do usuário, pois se deseja rotular os APKs que estão sendo baixados. Como já dito, nesse módulo é preciso respeitar algumas limitações diárias do VirusTotal. Quando a rotulação terminar de analisar um APK e processar os dados de rotulação, ela sinaliza para o módulo de geração que o dado está pronto para ser processado, assim como a extração.

Por fim, o módulo de geração permanece em um modo *wait* (espera) para verificar se ambos os dados de rotulação e geração para um mesmo APK estão prontos, se estiverem, o módulo pega as características que serão selecionadas e filtra o dado de rotulação no arquivo de análise, concatenando ambos e adicionando-os no *dataset*. Nessa etapa de geração, alguns passos abstraídos são realizados, como o tratamento dos dados.

4.6.1 Código Base

A ferramenta ADBuilder foi implementada por um código base em Python que fica responsável por gerenciar todo o processo de construção do *dataset*. É através dele que o usuário irá configurar e executar a ferramenta de acordo com suas necessidades. A Figura 10 mostra as possíveis opções de parâmetros para executar a ferramenta.

Figura 10 – Trecho de Código apresentando os parâmetros da ferramenta.

```
18  def parse_args(argv):
19      parser = argparse.ArgumentParser()
20      parser.add_argument('--file', type=str, help='TXT file with a list of APKs SHA256.')
21      parser.add_argument('--download', help='Download APK files.', action="store_true")
22      parser.add_argument('--n_parallel_download', '-npd', type=int, default=1, help='Number of Parallel Process for Dow
23      parser.add_argument('--extraction', help='APK Feature Extraction.', action="store_true")
24      parser.add_argument('--n_parallel_extraction', '-npe', type=int, default=1, help='Number of Parallel Process for F
25      parser.add_argument('--labelling', help='Virus Total Labelling.', action="store_true")
26      parser.add_argument('--vt_keys', type=str, help='TXT file with a VirusTotal\'s List of API Keys to Analysis.')
27      parser.add_argument('--building', help='Building Dataset.', action="store_true")
28      parser.add_argument('--delete', help='Delete All Previous Files.', action="store_true")
29
30      args = parser.parse_args(argv)
31      return args
32
33  def create_directories(_dirs):
34      for _dir in _dirs.keys():
35          Path(_dirs[_dir]).mkdir(parents=True, exist_ok=True)
36
37  def create_logs(_logs):
38      create_directories(_logs)
39
40  def create_queues(_queues):
41      create_directories(_queues)
```

Fonte: Autor (2023).

Conforme interpretado na Figura 10, o usuário poderá executar a ferramenta ADBuilder configurando alguns parâmetros, que serão apresentados a seguir:

- **—file** - O parâmetro *—file* recebe o nome do arquivo de texto que contém os SHA256 que identificam os aplicativos a serem processados. Os módulos de *download* e rotulação precisam receber esse arquivo para saber quais APKs que serão baixados e rotulados, respectivamente.
- **—download** - Quando esse parâmetro é ativado, a ferramenta ADBuilder é sinalizada que o módulo de *download* será ativado e executará o *download* dos aplicativos que estão listados no arquivo enviado pelo parâmetro *—file*.
- **—n_parallel_download** ou **-npd** - São parâmetros que recebem um número inteiro definido pelo usuário, que configura o número de processos que irão realizar o *download* dos APKs paralelamente. Se esse parâmetro não for ativado, por padrão é definido como 1.
- **—extraction** - Quando esse parâmetro é ativado, a ferramenta ADBuilder é sinalizada que o módulo de extração será executado e extrairá as características estáticas dos APKs que já foram coletados na fila de *download*.
- **—n_parallel_extraction** ou **-npe** - São parâmetros que recebem um número inteiro definido pelo usuário, que configura o número de processos que irão realizar a extração de características dos APKs paralelamente. Se esse parâmetro não for ativado, por padrão é definido como 1.
- **—labelling** - Quando esse parâmetro é ativado, a ferramenta ADBuilder é sinalizada que o módulo de rotulação será executado e rotulará os aplicativos que estão listados no arquivo enviado pelo parâmetro *–file*.
- **—vt_keys** - O parâmetro *—vt_keys* recebe o nome do arquivo de texto que contém a chave de API que será utilizada para acessar o serviço do VirusTotal para rotular os APKs. Esse parâmetro precisa ser ativado junto com o *—labelling* para o módulo de rotulação ser executado.
- **—building** - Quando esse parâmetro é ativado, a ferramenta ADBuilder é sinalizada que o módulo de geração será executado e irá construir o *dataset* com os arquivos dos módulos de extração e rotulação, já processados e prontos para serem utilizados.
- **—delete** - Quando esse parâmetro é ativado, a ferramenta exclui todos os arquivos gerados nas filas e logs dos módulos, processados em uma execução anterior.

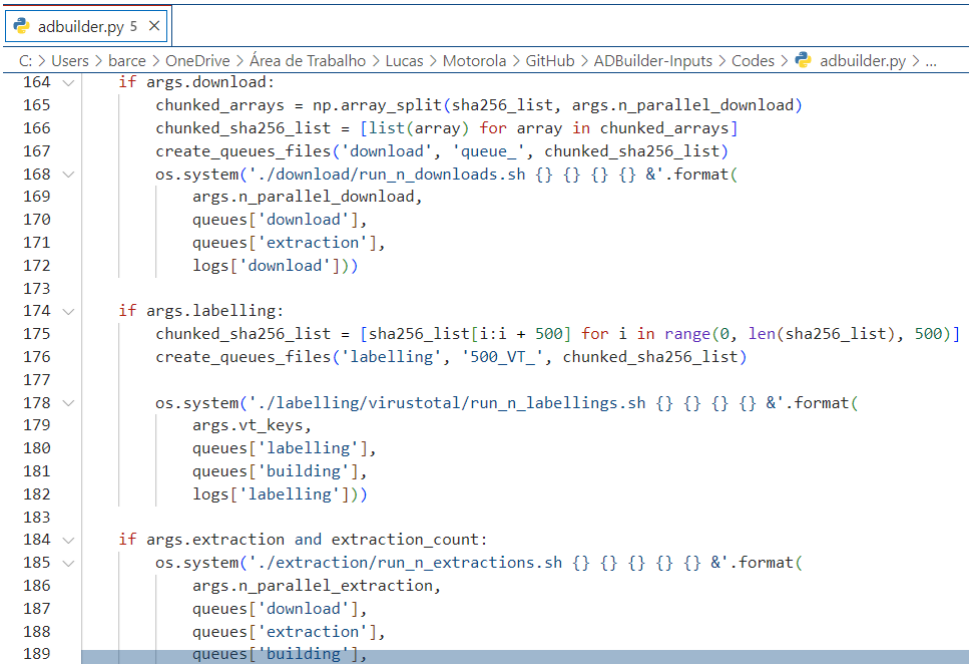
A documentação completa da ferramenta ADBuilder pode ser encontrada no repositório GitHub do projeto[43]. Nessa documentação, os usuários terão acesso a todas

[43]https://github.com/Overycall/ADBuilder-TCC

as dependências necessárias para a execução desta ferramenta, bem como um tutorial detalhado sobre como configurar e iniciar o ADBuilder. Além disso, a documentação inclui explicações claras sobre todos os parâmetros disponíveis na ferramenta, facilitando sua utilização e personalização conforme as necessidades de cada usuário.

Ainda tratando-se da Figura 10, é possível observar que o código *adbuilder.py* é responsável por preparar todo o ambiente para a execução da ferramenta, como por exemplo, criar as filas e os *logs* para serem utilizados por cada módulo.

Figura 11 – Trecho de Código Base em Python.



Fonte: Autor (2023).

Na Figura 11, é possível observar que através das condições do código é que a ferramenta controla quais módulos serão ativados, de acordo com os parâmetros que são configurados. Esse código cria as filas a serem utilizadas pelos módulos e o diretório dos *logs* estatísticos. Além disso, é responsável por exibir um relatório no terminal que ajuda o usuário acompanhar o progresso da construção do *dataset*. Ressaltando que o módulo de *download* e rotulação só podem funcionar se for passado um arquivo de texto, contendo os SHA256, enviado através do parâmetro —*file*. Em relação ao *download*, o código base recebe o número de processos paralelos que serão utilizados na coleta dos aplicativos através do repositório AndroZoo e, a partir disso, separa a lista de SHA256 em diversas partes, uma para cada processo, em seguida, executa o código *run_n_downloads.sh*.

4.6.2 Download

As Figuras 12 e 13, a seguir, apresentam os códigos que implementam o módulo de *download*.

Figura 12 – Trecho de Código *run_n_downloads* do Módulo de *Download*.

run_n_downloads.sh U

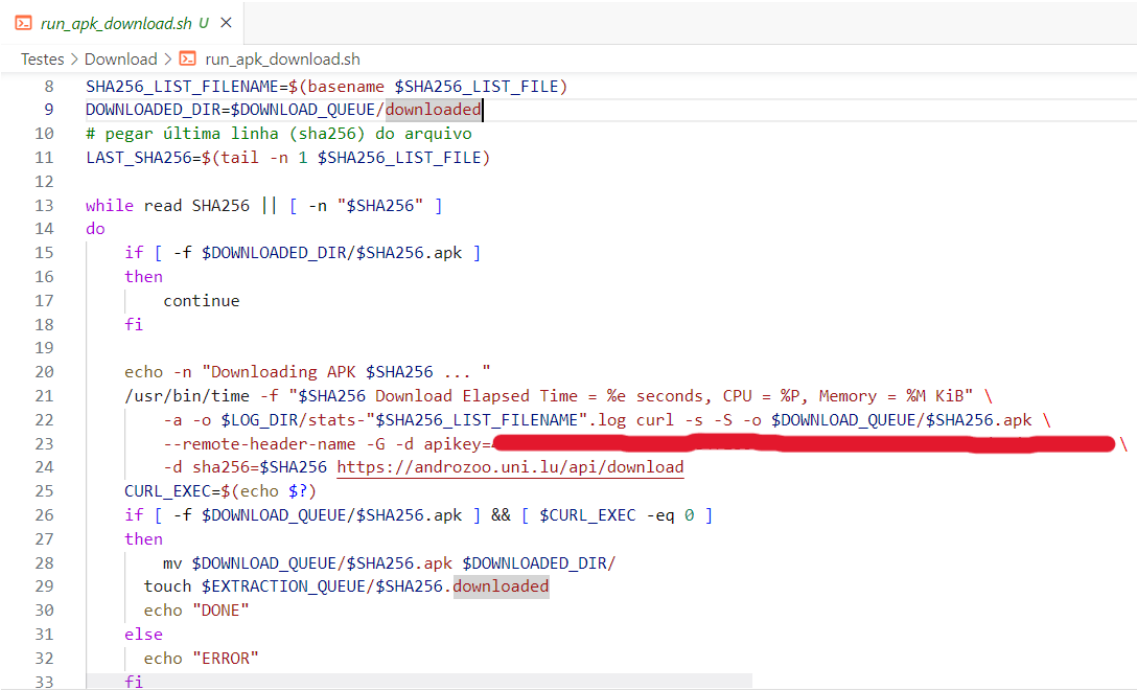
download > run_n_downloads.sh

```
9   DOWNLOADED_DIR=$DOWNLOAD_QUEUE/downloaded
10  [ -d $DOWNLOADED_DIR ] || { mkdir -p $DOWNLOADED_DIR; }
11  [ -f $DOWNLOAD_QUEUE/download.finished ] && { rm $DOWNLOAD_QUEUE/download.finished; }
12  [ -d $LOG_DIR ] || { mkdir -p $LOG_DIR; }
13
14  COUNTER=0
15  TS=$(date +%Y%m%d%H%M%S)
16
17  for SHA256_LIST_FILE in $DOWNLOAD_QUEUE/queue_*
18  do
19    [ -d $LOG_DIR/stats-$TS-$COUNTER ] || { mkdir -p $LOG_DIR/stats-$TS-$COUNTER; }
20    ./download/run_apk_download.sh $SHA256_LIST_FILE $DOWNLOAD_QUEUE $EXTRACTION_QUEUE $LOG_DIR/st
21      ((COUNTER++))
22  done
23
24  while [ 1 ]
25  do
26      COUNTER_FINISHED=$(find $DOWNLOAD_QUEUE -type f -name \*.finished | wc -l)
27      if [ $COUNTER_FINISHED -eq $COUNTER ]
28      then
29          kill `ps guaxwww | grep run_apk_download.sh | awk '{print $2}'` > /dev/null 2>&1
30      touch $DOWNLOAD_QUEUE/download.finished
31          exit
32    else
33      sleep 5
34      fi
```

Fonte: Autor (2023).

A Figura 12, apresenta o código em *Shell* que inicia o processo da ferramenta para executar o módulo de *download*. Para isso, é executado o código *run_apk_download* enviando a lista de SHA256, do processo em questão, para baixar os APKs. Após isso, o código executa um laço, verificando a cada 5 segundos se todos os aplicativos da lista foram coletados, se sim, encerra o processo e sinaliza que em sua fila o *download* foi concluído.

Figura 13 – Trecho de Código *run_apk_download* do Módulo de *Download*.



run_apk_download.sh U

Testes > Download > run_apk_download.sh

```
8   SHA256_LIST_FILENAME=$(basename $SHA256_LIST_FILE)
9   DOWNLOADED_DIR=$DOWNLOAD_QUEUE/downloaded
10  # pegar última linha (sha256) do arquivo
11  LAST_SHA256=$(tail -n 1 $SHA256_LIST_FILE)
12
13  while read SHA256 || [ -n "$SHA256" ]
14  do
15      if [ -f $DOWNLOADED_DIR/$SHA256.apk ]
16      then
17          continue
18      fi
19
20      echo -n "Downloading APK $SHA256 ... "
21      /usr/bin/time -f "$SHA256 Download Elapsed Time = %e seconds, CPU = %P, Memory = %M KiB" \
22          -a -o $LOG_DIR/stats-"$SHA256_LIST_FILENAME".log curl -s -S -o $DOWNLOAD_QUEUE/$SHA256.apk \
23          --remote-header-name -G -d apikey=[redacted] \
24          -d sha256=$SHA256 https://androzoo.uni.lu/api/download
25      CURL_EXEC=$(echo $?)
26      if [ -f $DOWNLOAD_QUEUE/$SHA256.apk ] && [ $CURL_EXEC -eq 0 ]
27      then
28          mv $DOWNLOAD_QUEUE/$SHA256.apk $DOWNLOADED_DIR/
29        touch $EXTRACTION_QUEUE/$SHA256.downloaded
30        echo "DONE"
31      else
32        echo "ERROR"
33      fi
```

Fonte: Autor (2023).

Observa-se na Figura 13 que o AndroZoo disponibiliza um comando *curl* para ser utilizado para baixar os APKs através de sua API. O parâmetro *APIKEY* é referente a chave de API liberada ao se cadastrar no site do AndroZoo[44]. Lembrando que cada usuário terá que ter sua própria chave de API para poder baixar os aplicativos do AndroZoo. A chave utilizada nos testes dessa pesquisa é exclusiva do projeto. Já o parâmetro *SHA256* é o resumo criptográfico que identifica unicamente o aplicativo no banco de dados do repositório e o transfere para a máquina local. Quando um APK é baixado, o código sinaliza o módulo de extração que o arquivo está pronto para ser utilizado. Além disso, ainda é possível observar na Figura 13 que o comando *curl* é executado para realizar a coleta do APK, juntamente com a biblioteca *time* do sistema operacional, para evidenciar as métricas de execução do *download*, onde são relatadas em um arquivo na fila de *logs* do módulo.

4.6.3 Extração

As Figuras 14, 15 e 16, a seguir, apresentam os códigos que implementam o módulo de extração.

[44]https://androzoo.uni.lu/api_doc

Figura 14 – Trecho do código *run_n_extractions* do Módulo de Extração.

run_n_extractions.sh U

extraction > run_n_extractions.sh

```
16  COUNTER=1
17  TS=$(date +%Y%m%d%H%M%S)
18
19  for NEXT in $(seq 1 $N_PARALLEL_EXTRACTORS)
20  do
21      [ -d $LOG_DIR/stats-$TS-$COUNTER ] || { mkdir -p $LOG_DIR/stats-$TS-$COUNTER; }
22      bash -x ./extraction/run_apk_extraction.sh $DOWNLOAD_QUEUE $EXTRACTION_QUEUE $BUILDING_QUE
23      ((COUNTER++))
24  done
25
26  while [ 1 ]
27  do
28      # contabilizar arquivos .downloaded e .extracted
29      EXT_DOWNLOAD=$(find $DOWNLOAD_QUEUE/downloaded -type f -name \*.apk | wc -l)
30      EXT_COUNT=$(find $EXTRACTION_QUEUE/extracted -type f -name \*.json | wc -l)
31
32      # verifica se o download foi finalizado
33      if [ -f $DOWNLOAD_QUEUE/download.finished ] && [ $EXT_DOWNLOAD -eq $EXT_COUNT ]
34      then
35          touch $EXTRACTION_QUEUE/extraction.finished
36          kill `ps guaxwww | grep run_apk_extraction.sh | awk '{print $2}'` > /dev/null 2>&1
37          exit
38      else
39        sleep 5
40      fi
41  done
```

Fonte: Autor (2023).

A Figura 14 apresenta o código *run_n_extractions*, implementado em *Shell*, que executa o código *run_apk_extraction.sh* repetidamente N vezes. O valor de N é determinado pelo usuário e representa o número de processos do módulo de extração responsáveis por realizar a engenharia reversa nos APKs e extrair suas características. Após cada iteração, o código verifica se o número de aplicativos processados pelo módulo de *download* e extração é igual. Essa verificação ocorre a cada 5 segundos, enquanto o código também monitora a fila de *download* para determinar se todos os aplicativos foram baixados. Quando o número de APKs baixados é igual ao número de APKs extraídos e não há mais aplicativos na fila de *download*, o módulo de extração encerra todos os seus processos. Esse mecanismo garante a conclusão adequada do processo de extração e sincronização entre os módulos envolvidos.

Figura 15 – Trecho do código *run_apk_extraction* do Módulo de Extração.

run_apk_extraction.sh U

extraction > run_apk_extraction.sh

```
24  APK_FILENAME=$(basename $FILE .downloaded)
25
26  if [ -f $EXTRACTED_DIR/$APK_FILENAME.json ]
27    then
28      continue
29  fi
30
31  # pega o LOCK do APK. se o LOCK ja existir, pula para o proximo APK.
32  if { set -C; 2>/dev/null > $EXTRACTION_QUEUE/$APK_FILENAME.lock; }; then
33    trap "rm -f $EXTRACTION_QUEUE/$APK_FILENAME.lock" EXIT
34  else
35    continue # arquivo de LOCK ja existe. vai para proximo APK.
36  fi
37
38  mv $EXTRACTION_QUEUE/$APK_FILENAME.downloaded $EXTRACTION_QUEUE/$APK_FILENAME.extracting
39
40  echo -n "Starting Processing APK File $APK_FILENAME ... "
41  # extrai as caracteristicas do APK e gera estatisticas
42  /usr/bin/time -f "$APK_FILENAME Extraction Elapsed Time = %e seconds, CPU = %P, Memory = %M KiB" \
43    -a -o $LOGS_DIR/stats-extraction-$COUNTER.log python3 extraction/extract_apk_features.py \
44    --apk $DOWNLOADED_DIR/$APK_FILENAME.apk --outdir $EXTRACTION_QUEUE --logdir $LOGS_DIR
45
46  if [ -f $EXTRACTION_QUEUE/$APK_FILENAME.json ]
47  then
48    rm -f $EXTRACTION_QUEUE/$APK_FILENAME.extracting
49    rm -f $EXTRACTION_QUEUE/$APK_FILENAME.lock
50    # sinaliza os processos de building que o CSV ja foi todo gravado
51    touch $BUILDING_QUEUE/$APK_FILENAME.extracted
52    mv $EXTRACTION_QUEUE/$APK_FILENAME.json $EXTRACTED_DIR/
53    echo "DONE"
54  fi
55  sleep 10
```

Fonte: Autor (2023).

Conforme descrito na Figura 15, observa-se uma seção do código *run_apk_extraction* implementado em *Shell*. Nesse trecho, a cada 10 segundos, é verificado se existem APKs que já foram coletados. Quando um aplicativo é completamente baixado, o processo de extração gera um arquivo de bloqueio (*lock*) para indicar que esse arquivo está sendo processado e evitar conflitos de concorrência entre os processos. Em seguida, é invocado o código *extract_apk_features* implementado em Python, responsável por extrair as características do APK. Esse código é executado juntamente com a biblioteca *time* do sistema operacional, que calcula as métricas de execução do processo. Após a extração das características do arquivo, elas são armazenadas em um arquivo JSON e o bloqueio do arquivo é liberado. Esse fluxo de processamento garante uma extração segura e organizada das características dos APKs.

Figura 16 – Trecho de Código em Python do Módulo de Extração.

```python
def extract_features(args):
    logging.basicConfig(format = '%(name)s - %(levelname)s - %(message)s')
    global logger
    global sha256
    logger = logging.getLogger('Extraction')
    try:
        f = open(args.apk, 'rb')
        contents = f.read()
    except e:
        info = colored(e, 'red')
        logger.exception(info)
        exit(1)

    sha256 = hashlib.sha256(contents).hexdigest()
    sha256 = sha256.upper()
    app, d, dx = AnalyzeAPK(args.apk)

    try:
        app_name = app.get_app_name()
    except:
        app_name = 'Not Found'
        info_warning_list('App Name Not Found')

    package = app.get_package()
    target_sdk = app.get_effective_target_sdk_version()
    min_sdk = app.get_min_sdk_version()
    try:
        permissions = app.get_permissions()
        permissions = [p.split('.')[-1] for p in permissions]
    except:
        permissions = info_warning_list('Could Not Extract Permissions')

```

Fonte: Autor (2023).

No trecho de código mostrado na Figura 16, é apresentado o método de extração das características mais básicas da ferramenta Androguard, como nome e pacote do APK, versões de API, permissões, intenções e componentes (i.e., serviços, atividades, receptores e provedores). Para extrair as chamadas de API e *opcodes* utiliza-se outros métodos disponibilizados pela ferramenta, pois a forma de extraí-los é diferente, onde é preciso percorrer o arquivo *DEX* da aplicação e extrair os métodos externos (i.e., chamadas de API) e os métodos em baixo nível que os invocam (i.e., *opcodes*). Além disso, as chamadas de API são filtradas através de um grupo seleto de pacotes oficiais do Android, com o objetivo de introduzir no *dataset* um número reduzido, porém eficiente de chamadas de API, pois o número de ocorrências totais em um APK pode resultar em um *dataset* com muitas colunas, necessitando de um *hardware* mais potente para poder manipular um grande conjunto de dados. Porém, a extração crua das chamadas de API é armazenada em um arquivo de texto para que possa ser utilizada para estudos posteriores ou para construir um *dataset* personalizado de chamadas de API. O arquivo que contém o grupo seleto de pacotes de chamadas de API e uma lista de intenções para ser filtradas, é o *constants.py*.

4.6.4 Rotulação

O módulo de rotulação consome a API do VirusTotal, o qual realiza uma busca em seu banco de dados pela última análise realizada para o APK em questão, e então, retorna esses dados para a ferramenta ADBuilder. Essa, por sua vez, analisa o arquivo retornado pela API e filtra o dado de rotulação, isto é, a quantidade de *scanners* que identificaram e rotularam o APK como malicioso. O módulo de rotulação foi implementado por códigos em Python e *Shell Script*. As Figuras 17 e 18, a seguir, apresentam trechos de código em *Shell* e Python, respectivamente, que resultam no módulo de rotulação.

Figura 17 – Trecho de Código em *Shell* do Módulo de Rotulação.

run_apk_labelling.sh

labelling > virustotal > run_apk_labelling.sh

```
15  while read SHA256 || [ -n "$SHA256" ]
16  do
17      if [ -f $LABELED_DIR/$SHA256.csv ]
18      then
19          continue
20      fi
21
22      echo -n "Downloading JSON $SHA256 ... "
23      echo -e "\nUsed VT API Key: $API_KEY"
24
25      /usr/bin/time -f "$SHA256 Elapsed Time of VT Analysis = %e seconds, CPU = %P, Memory = %M KiB" \
26          -a -o $LOG_DIR/stats-"$SHA256_LIST_FILENAME".log \
27          python3 labelling/virustotal/label.py \
28          --sha256 $SHA256 --vt_key $API_KEY --outdir $LABELLING_QUEUE
29
30      if [ -f $LABELED_DIR/$SHA256.csv ]
31      then
32          touch $BUILDING_QUEUE/$SHA256.labeled
33      fi
34
35      echo -e "Finished $SHA256 Label!!!\n"
36
37      # verificar se o último APK do arquivo já foi baixado
38      if [ -f $LABELED_DIR/$LAST_SHA256.csv ] || [ -f $LABELLING_QUEUE/Errors/$LAST_SHA256.json ]
39      then
40          mv $LABELLING_QUEUE/$SHA256_LIST_FILENAME $LABELLING_QUEUE/$SHA256_LIST_FILENAME.finished
```

Fonte: Autor (2023).

De acordo com a Figura 17, a implementação em *Shell* lê um arquivo contendo os SHA256 dos aplicativos e verifica se existe o arquivo de extração do APK em questão, se não existir, é porque o mesmo ainda não foi processado. Quando o módulo começar a processar o APK, ele invocará o código em Python, chamado *label.py*, para realizar a consulta na API do VirusTotal, juntamente com a biblioteca *time* do sistema operacional, para calcular as métricas do processo de rotulação. Quando a aplicação for processada e os arquivos gerados, o módulo de rotulação sinalizará o módulo de geração que os arquivos estão prontos para serem utilizados.

Figura 18 – Trecho de Código em Python do Módulo de Rotulação.

```
label.py 2 ×
labelling > virustotal > label.py > ...
102     if json_data:
103         try:
104             last_analysis_date = int(json_data['data']['attributes']['last_analysis_date'])
105             date = datetime.datetime.fromtimestamp(last_analysis_date)
106             human_readable_date = date.strftime('%Y-%m-%d %H:%M:%S')  # format the date as desired
107             print(f'Last Analysis: {human_readable_date}')
108 
109             app = json_data['data']['attributes']
110             last_stats = app['last_analysis_stats']
111             app_data = {'sha256': sha256,
112                         'malicious': last_stats['malicious'],
113                         'undetected': last_stats['undetected'],
114                         'harmless': last_stats['harmless'],
115                         'suspicious': last_stats['suspicious'],
116                         'failure': last_stats['failure'],
117                         'unsupported': last_stats['type-unsupported'],
118                         'timeout': last_stats['timeout'],
119                         'confirmed_timeout': last_stats['confirmed-timeout']}
120             save_as_json(sha256, json_data, os.path.join(outdir, 'labeled'))
121             labelling_data = pd.DataFrame([[app_data['sha256'],app_data['malicious'], last_analysis_date]], columns=['
122             sha256_csv = os.path.join(outdir, 'labeled', f'{sha256}.csv')
123             labelling_data.to_csv(sha256_csv, index = False)
124             with open('log.txt', 'a') as f:
125                 f.write(f'{sha256}, {vt_key}\n')
126         except JSONDecodeError as errd:
127             e = colored(f'JSON Decoder Error: {errd}', 'red')
```

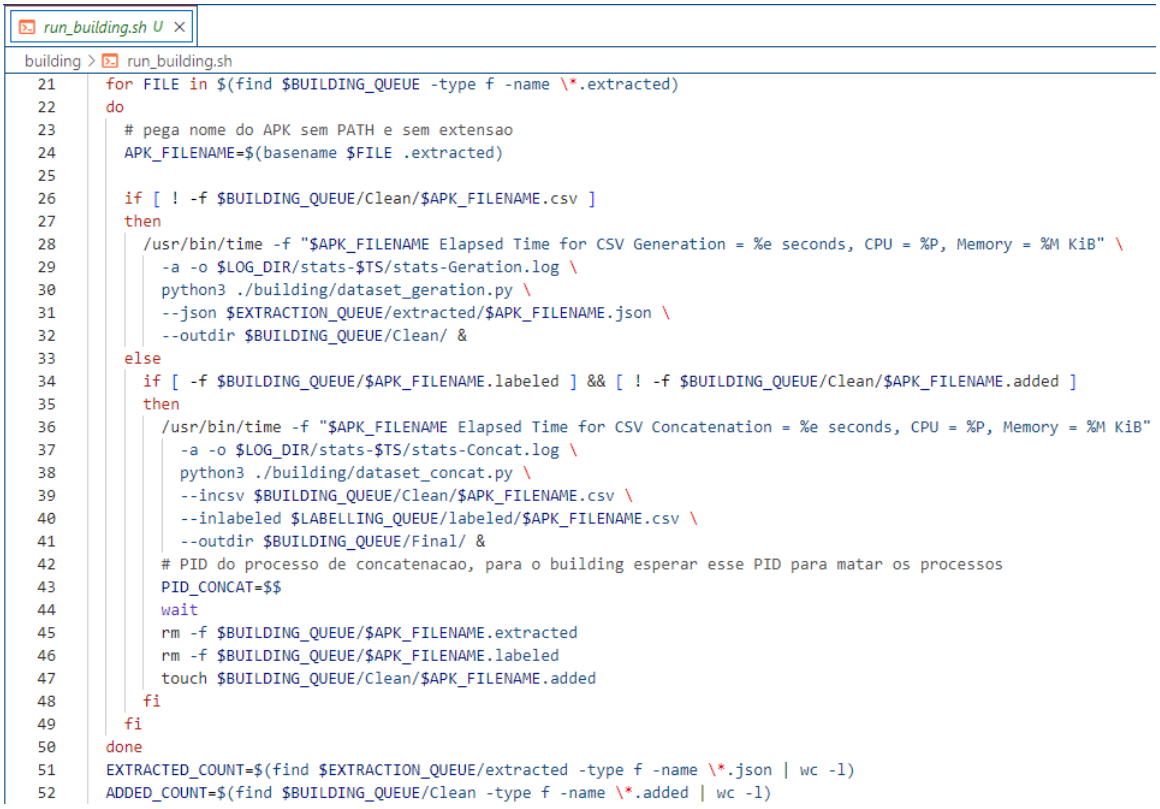
Fonte: Autor (2023).

A Figura 18 apresenta o código em Python que implementa o módulo de rotulação, sendo desta forma, responsável pela integração do serviço do VirusTotal com o módulo de rotulação da ADBuilder. Esse código é invocado com três parâmetros: (a) o SHA256 do aplicativo que se deseja rotular; (b) a chave de API para utilizar o serviço online do VirusTotal; e (c) o diretório de saída (i.e., a fila do módulo de rotulação) onde os arquivos serão gerados. Após o código processar o APK, assim como é gerado o arquivo de rotulação, é gerado um arquivo .csv contendo dados em relação a rotulação do aplicativo (e.g., a data da última análise do APK realizada no banco de dados do VirusTotal). Lembrando que será disponibilizada uma chave de API para utilizar o serviço do VirusTotal. Porém, é importante cada usuário criar sua chave, ao se cadastrar no site, devido as limitações diárias.

4.6.5 Geração

O módulo de geração é o responsável por extrair os dados dos módulos de extração e rotulação, manipulá-los, e com isso, construir o *dataset*. As Figuras 19, 20 e 21, apresentam os códigos, desenvolvidos em Python e *Shell*, que implementam o módulo de geração, da ferramenta.

Figura 19 – Trecho de Código em *Shell* do Módulo de Geração.

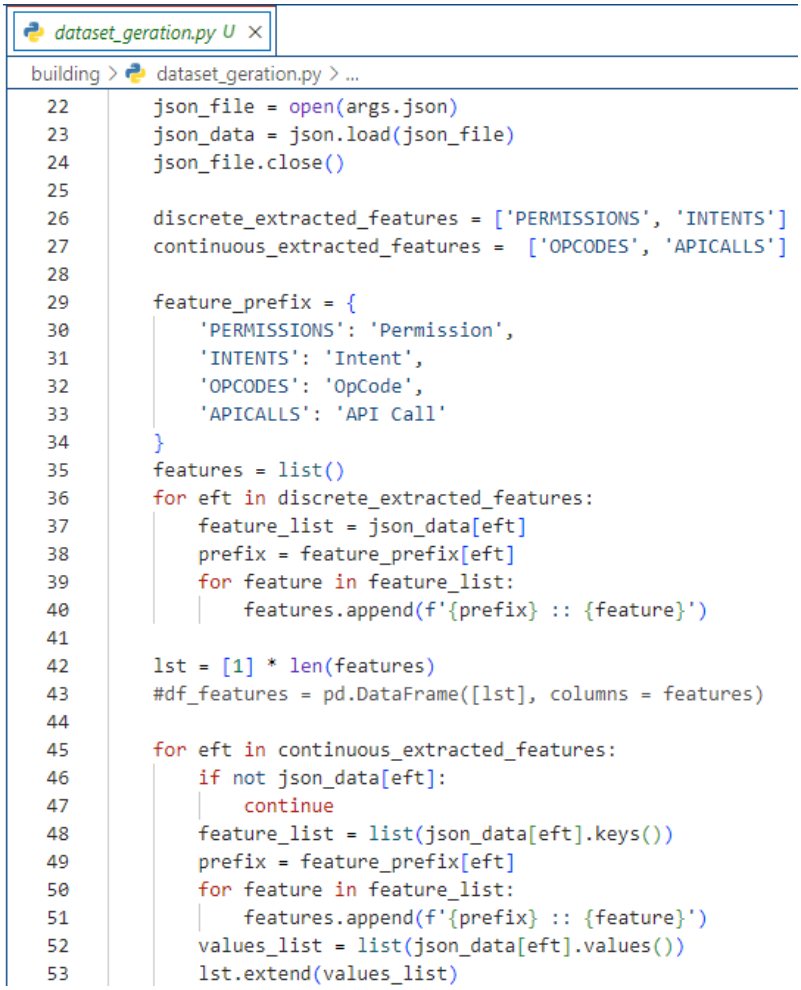


```
run_building.sh U
building > run_building.sh
21  for FILE in $(find $BUILDING_QUEUE -type f -name \*.extracted)
22  do
23    # pega nome do APK sem PATH e sem extensao
24    APK_FILENAME=$(basename $FILE .extracted)
25
26    if [ ! -f $BUILDING_QUEUE/Clean/$APK_FILENAME.csv ]
27    then
28      /usr/bin/time -f "$APK_FILENAME Elapsed Time for CSV Generation = %e seconds, CPU = %P, Memory = %M KiB" \
29        -a -o $LOG_DIR/stats-$TS/stats-Geration.log \
30        python3 ./building/dataset_geration.py \
31        --json $EXTRACTION_QUEUE/extracted/$APK_FILENAME.json \
32        --outdir $BUILDING_QUEUE/Clean/ &
33    else
34      if [ -f $BUILDING_QUEUE/$APK_FILENAME.labeled ] && [ ! -f $BUILDING_QUEUE/Clean/$APK_FILENAME.added ]
35      then
36        /usr/bin/time -f "$APK_FILENAME Elapsed Time for CSV Concatenation = %e seconds, CPU = %P, Memory = %M KiB"
37          -a -o $LOG_DIR/stats-$TS/stats-Concat.log \
38          python3 ./building/dataset_concat.py \
39          --incsv $BUILDING_QUEUE/Clean/$APK_FILENAME.csv \
40          --inlabeled $LABELLING_QUEUE/labeled/$APK_FILENAME.csv \
41          --outdir $BUILDING_QUEUE/Final/ &
42        # PID do processo de concatenacao, para o building esperar esse PID para matar os processos
43        PID_CONCAT=$$
44        wait
45        rm -f $BUILDING_QUEUE/$APK_FILENAME.extracted
46        rm -f $BUILDING_QUEUE/$APK_FILENAME.labeled
47        touch $BUILDING_QUEUE/Clean/$APK_FILENAME.added
48      fi
49    fi
50  done
51  EXTRACTED_COUNT=$(find $EXTRACTION_QUEUE/extracted -type f -name \*.json | wc -l)
52  ADDED_COUNT=$(find $BUILDING_QUEUE/Clean -type f -name \*.added | wc -l)
```

Fonte: Autor (2023).

A Figura 19, apresenta um trecho do código *run_building*, implementado em *Shell*, que verifica na fila de geração se existe algum arquivo vindo do módulo de extração que esteja pronto para ser utilizado. Quando o arquivo é encontrado, então é verificado se esse arquivo já foi processado pelo código *dataset_geration.py*, que realiza um tratamento nos dados e os prepara para serem inclusos no *dataset* em construção. Se o arquivo não foi processado, então é invocado o código *dataset_geration.py* para tratar o arquivo. O código é executado juntamente com o comando *time*, do sistema operacional, para gerar as métricas de execução do processo. No caso do arquivo ter sido processado, então é verificado se os dados de rotulação desse APK estão prontos para serem utilizados, se sim, então é invocado o código *dataset_concat.py*, juntamento com o comando *time*, para concatenar os dados de extração e rotulação do aplicativo, e adicioná-los ao *dataset*. O código implementado em *Shell* fica responsável por encerrar os processos do módulo de geração quando a ferramenta incluir no *dataset* todos os aplicativos baixados pelo módulo de *download*. Para isso, é necessário que o módulo de *download* sinalize para o módulo de geração que todos os APKs foram baixados, pois pode ocorrer de um processo da extração estar esperando um aplicativo ser baixado.

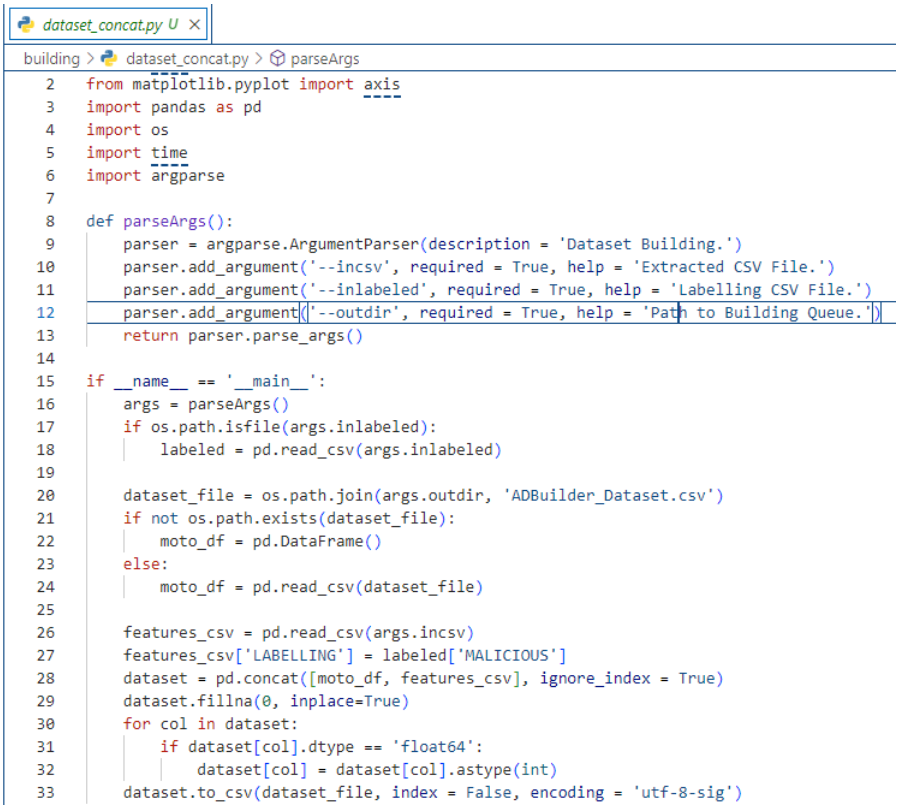
Figura 20 – Trecho do Código *dataset_geration.py* do Módulo de geração.



Fonte: Autor (2023).

No trecho de código observado na Figura 20, apresenta o código *dataset_geration*, implementado em Python, que abre o arquivo JSON que contém as características estáticas extraídas do aplicativo e realiza uma separação entre características discretas e contínuas. Logo após, para cada recurso, é inserido um prefixo para auxiliar no entendimento das categorias das características. Em seguida, todas as características são consideradas como ocorrências, então para cada uma é adicionado o valor "1" (um), indicando que há a ocorrência dessa característica no APK. Por fim, o código cria um *dataset* (i.e., em CSV) com uma única linha, representando unicamente o aplicativo em questão. Com isso, só faltará os dados de rotulação do APK, para que o mesmo seja incluso no *dataset*.

Figura 21 – Trecho do Código *dataset_concat.py* do Módulo de geração.



```
dataset_concat.py U
building > dataset_concat.py > parseArgs
 2  from matplotlib.pyplot import axis
 3  import pandas as pd
 4  import os
 5  import time
 6  import argparse
 7
 8  def parseArgs():
 9      parser = argparse.ArgumentParser(description = 'Dataset Building.')
10      parser.add_argument('--incsv', required = True, help = 'Extracted CSV File.')
11      parser.add_argument('--inlabeled', required = True, help = 'Labelling CSV File.')
12      parser.add_argument('--outdir', required = True, help = 'Path to Building Queue.')
13      return parser.parse_args()
14
15  if __name__ == '__main__':
16      args = parseArgs()
17      if os.path.isfile(args.inlabeled):
18          labeled = pd.read_csv(args.inlabeled)
19
20      dataset_file = os.path.join(args.outdir, 'ADBuilder_Dataset.csv')
21      if not os.path.exists(dataset_file):
22          moto_df = pd.DataFrame()
23      else:
24          moto_df = pd.read_csv(dataset_file)
25
26      features_csv = pd.read_csv(args.incsv)
27      features_csv['LABELLING'] = labeled['MALICIOUS']
28      dataset = pd.concat([moto_df, features_csv], ignore_index = True)
29      dataset.fillna(0, inplace=True)
30      for col in dataset:
31          if dataset[col].dtype == 'float64':
32              dataset[col] = dataset[col].astype(int)
33      dataset.to_csv(dataset_file, index = False, encoding = 'utf-8-sig')
```

Fonte: Autor (2023).

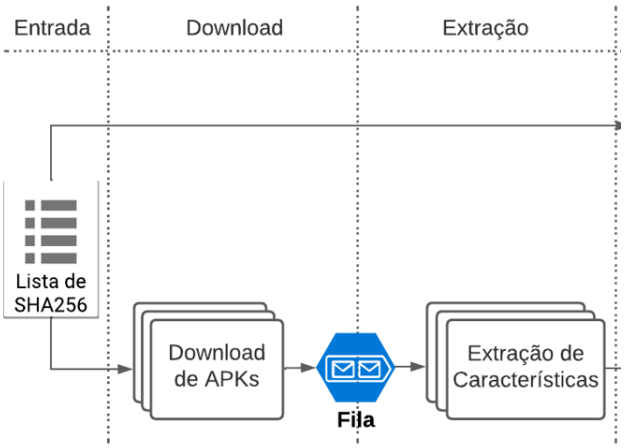
A Figura 21, apresenta o código *dataset_concat* implementado em Python, que é responsável por criar o *dataset* (i.e., em CSV) com o nome *"ADBuilder_Dataset"*, se não existir. Caso o conjunto de dados exista, o código apenas lê seus dados e adiciona uma nova amostra. Após isso, altera o nome da coluna *"MALICIOUS"* para *"LABELLING"*, do arquivo vindo do código *dataset_geration*, o qual contém os dados de rotulação (i.e., quantidade de *scanners* do VirusTotal que identificaram o APK como malicioso). Em seguida, preenche todos os valores nulos (i.e., NaN), com o valor "0" (zero), para indicar que não existe a ocorrência da característica no aplicativo. Estes valores nulos surgem quando o arquivo vindo do código *dataset_geration.py*, que contém os valores "1" indicando a ocorrência das características, for concatenado ao *dataset* final. Com essa concatenação, as colunas com mesmo nome em ambos arquivos mantém o valor, porém as colunas que estão presentes somente no *dataset* final, são geradas como nulas, indicando que não há a ocorrência das características no APK. Por fim, é realizada uma verificação para analisar o tipo de cada coluna do conjunto de dados. Se o tipo for *float64* (i.e., ponto flutuante), a coluna é convertida para inteiro, assim o tamanho do *dataset* pode ser diminuído em até 3 (três) vezes, levando em consideração que um número em ponto flutuante poderá conter até 3 caracteres a mais que um número inteiro (e.g., 1.00).

É importante ressaltar que o processo de concatenação é realizado sequencialmente, executado por apenas um processo, no qual precisa tanto dos dados de rotulação, vindos do módulo de rotulação, quanto do arquivo CSV com os dados tratados, vindo do código *dataset_geration.py*, presente na pasta chamada *"Clean"*, na fila do módulo de geração.

4.7 Resultados e Discussões

Em relação à arquitetura completa da ADBuilder, proposta em um momento inicial no Capítulo 4, os primeiros dois módulos (i.e., *download* e extração) foram implementados como parte do processo de validação da ferramenta. Durante esse período, foram conduzidos testes unitários e de integração para verificar o funcionamento desses módulos, tanto de forma individual quanto em conjunto.

Figura 22 – Arquitetura dos Dois Primeiros Módulos.



Fonte: Autor (2023).

O teste de extração de características mostrou ser rápido e eficiente, focando em extrair as características para realizar análises estáticas posteriormente. A análise estática e a análise dinâmica são dois métodos diferentes para analisar as características de um aplicativo no sistema operacional Android. A análise estática é um processo de análise no qual o aplicativo é examinado sem ser executado. Isso é feito geralmente através da desmontagem do arquivo APK (pacote de aplicativo Android) e análise dos arquivos e recursos contidos nele. A análise estática é geralmente mais rápida e mais fácil de realizar do que a análise dinâmica, pois não requer a execução do aplicativo. Além disso, é mais

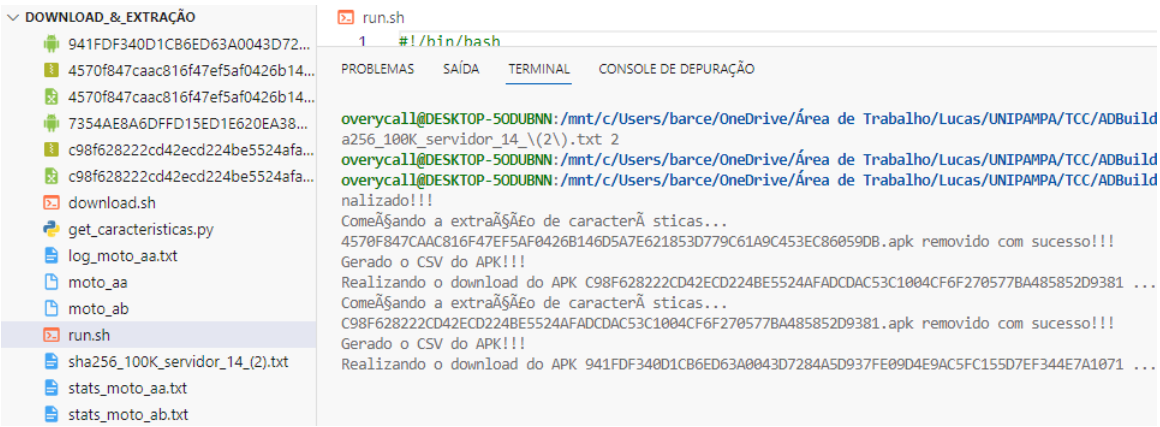
seguro, pois reduz o risco de causar danos ao dispositivo ou aos dados do usuário.

A análise dinâmica, por outro lado, é um processo de análise no qual o aplicativo é executado e monitorado enquanto está rodando. Isso permite a análise das ações e comportamentos do aplicativo em tempo real, incluindo acesso às redes, acesso ao sistema de arquivos e interação com outros aplicativos. A análise dinâmica é geralmente mais precisa do que a análise estática, pois permite ver o aplicativo em ação. No entanto, a realização desse procedimento é geralmente mais demorada e complexa, uma vez que demanda a configuração de um ambiente de teste e a monitoração contínua do aplicativo durante sua execução.

Em geral, a análise estática e a análise dinâmica são duas técnicas complementares que podem ser usadas juntas para obter uma visão mais completa das características de um aplicativo. Porém, para esse estudo foi elencado a utilização da análise estática devido a disponibilização de recursos para construir a ferramenta, menor risco e a simplicidade para gerar grandes volumes de dados.

A Figura 23, a seguir, apresenta os resultados da integração dos módulos de *download* e extração, utilizados em um servidor Linux Ubuntu 20.04. Observa-se no *print* da tela do terminal que a ferramenta realiza o *download* dos APKs e em seguida extrai suas características. O módulo de extração extrai 13 tipos de características diferentes: nome e pacote do APK, versões de API (recomendada e mínima), SHA256, permissões, chamadas de API, intenções, *opcodes*, atividades, serviços, provedores e receptores.

Figura 23 – Resultado da integração dos Módulos de *Download* e Extração.



Fonte: Autor (2023).

Uma das dificuldades encontradas na implementação foi em estudar e selecionar os métodos mais adequados para extrair as características. Além disso, as chamadas de API são as características que possuem o maior número de ocorrências nos aplicativos

quando comparadas com outras características, como já dito na seção 4.6. Ademais, outra dificuldade encontrada na implementação foi em selecionar quais chamadas de API deveriam ser filtradas. Com isso, foi realizado um levantamento para verificar quais chamadas de API são oficiais do Android e quais são do Java, as quais foram armazenadas em um arquivo disponível no repositório do projeto[45].

Ao final, foi possível concluir que todas as ferramentas necessárias foram testadas e implementadas para os dois primeiros módulos (i.e., *download* e extração) da ferramenta ADBuilder. A integração dessas ferramentas (i.e., AndroZoo e Androguard) permitiu o *download* do APK e a extração de suas características estáticas, armazenando seus dados em um arquivo JSON e salvando em um arquivo de texto a extração crua das chamadas de API (i.e., sem nenhum tipo de filtragem de dados), conforme é possível observar através da Figura 23. A Figura, a seguir, apresenta um exemplo de um arquivo JSON gerado pelo módulo de extração, mostrando algumas das características extraídas de um aplicativo, sendo elas: SHA256, nome, pacote, versões de API, listas de permissões, intenções, atividades, e demais categorias.

Figura 24 – Exemplo de um arquivo JSON contendo uma porção das características extraídas.


```json
{
    "SHA256": "0000CF055A25514C0C7B999B5AF216DD314C496F8C3B067AC314C4D63292B87E",
    "APP_NAME": "Almas del Purgatorio",
    "PACKAGE": "com.wilsonzuluaga.oracionalasalmasdelpurgatorioenaudioytexto",
    "TARGET_API": 22,
    "MIN_API": "14",
    "PERMISSIONS": [
        "ACCESS_WIFI_STATE",
        "VIBRATE",
        "INTERNET",
        "READ_EXTERNAL_STORAGE",
        "ACCESS_NETWORK_STATE"
    ],
    "INTENTS": [
        "MAIN",
        "LAUNCHER"
    ],
    "ACTIVITIES": [
        "com.appybuilder.wzuluaga82.Oracion_a_las_almas_del_purgatorio_en_audio_y_texto.Screen1",
        "com.google.android.gms.ads.AdActivity",
        "com.google.appinventor.components.runtime.ListPickerActivity"
    ],
    "SERVICES": [],
    "RECEIVERS": [],
    "PROVIDERS": [],
    "OPCODES": {
```


Fonte: Autor (2023).

Analisando a Figura 24, é possível perceber que os metadados como SHA256,

[45] https://github.com/Overycall/ADBuilder-TCC/blob/main/extraction/constants.py

nome, pacote e versões de API são dados de valores únicos. Porém, características como permissões, intenções e chamadas de API são extraídas e armazenadas em um vetor, devido a simplicidade de agrupação e manipulação desses dados.

A Figura 25, representa um arquivo JSON retornado pela análise da API do VirusTotal, que contém informações sobre características, *Strings* (e.g., URLs), metadados gerais e dados dos *scanners* sobre a análise do APK. O VirusTotal possui mais de 60 *scanners*, e para cada um, é retornado o resultado de sua análise sobre o APK em questão, informando se foi identificado alguma anomalia. No final, todos esses resultados são agrupados e apresentados ao usuário no final, como é possível observar na Figura 25. Por exemplo, o atributo *malicious* representa a quantidade de *scanners* que identificaram o aplicativo como malicioso, ao contrário do atributo *undetected*, que representa a quantidade de *scanners* que não identificaram nenhuma atividade suspeita na aplicação. O número apresentado no atributo *type-unsupported* indica que os *scanners* não suportam o tipo de análise de arquivo APK.

Figura 25 – Exemplo de um arquivo JSON contendo os dados da análise do VirusTotal.

Fonte: Autor (2023).

A Figura 26, apresenta o arquivo CSV contendo todas as características (e metadados) de um aplicativo, que são adicionados no *dataset* em construção. Os dados desse arquivo já foram processados e formatados para serem incluídos no conjunto de

dados. É possível observar que as características, como permissões, foram inseridas com o valor 1 (um) para indicar que há a ocorrência da mesma no APK, sendo que isso acontece para todas as listas de características, como atividades, intenções, *Opcodes* e chamadas de API, pois conforme descrito na Figura 24, as características são agrupadas em um vetor, onde são destrinchadas no módulo de geração.

Figura 26 – Exemplo de um arquivo CSV já processado pelo módulo de geração.

SHA256	APP_NAME	PACKAGE	TARGET_API	MIN_API	Permission :: INTERNET	Permission :: VIBRATE	Permission :: ACCESS_WIFI_STATE
14C496F8C3B06...	Almas del Purgatorio	com.wilsonzuluaga.oracionalasalmasdelpurgatori...	22	14	1	1	1

Fonte: Autor (2023).

Segundo a Figura 26, as características recebem um prefixo (e.g., “Permissions ::”) para facilitar no entendimento das nomenclaturas dos recursos para o usuário, visto que já foi presenciado em muitos *datasets* da literatura, a ausência da identificação das categorias das características.

O *dataset* gerado pela ADBuilder, conforme apresentado na Figura 27, é composto por um conjunto de 350 APKs, totalizando 18979 características extraídas. Esse conjunto abrange uma variedade de metadados, como SHA256, nome e pacote do APK, além de versões de API. Além disso, o *dataset* inclui um conjunto abrangente de características, como permissões, intenções, *opcodes* e chamadas de API. Notavelmente, foram identificadas 440 permissões, 1166 intenções, 17144 chamadas de API e 223 *opcodes*.

Figura 27 – *Dataset* de 350 APKs gerado pela ADBuilder.

SHA256	APP_NAME	PACKAGE	TARGET_API	MIN_API	Permission :: WRITE_EXTERNAL_STORAGE	Permission ACCESS_NETWORK_STAT
903F371...	Traffic Tour Racer	com.traffic.tour.racer.pro	28	16	1	
202687A...	Wired	com.magmamobile.game.Wired.free	26	5	0	
16F01CD...	AIRS	au.com.safeware.airs	30	21	1	
D63BF35...	Colorful Flame Skull Theme	com.colorful.flame.skull.theme	26	15	1	
377816C...	小说大全	com.tadu.android.androidread	22	14	1	
11E8FB0...	音标学习	air.com.aaaedu.doubleenphonetic	27	14	0	
F7AC571...	嗖56	cn.net.cnea.transportcapacitycooldemo	26	17	1	
F8C3B06...	Almas del Purgatorio	com.wilsonzuluaga.oracionalasalmasdelpurgatori...	22	14	0	
78B3CC0...	Medicine app	com.medicineapp	28	24	0	
EF604AC...	ibarrel爱杯	cn.com.ibarrel.hugs	25	17	1	

Fonte: Autor (2023).

Salienta-se que, mesmo com um número relativamente pequeno de apenas 350 APKs, o *dataset* gerado pela ADBuilder apresenta uma quantidade significativa de características, ultrapassando a marca de 18 mil. Essa diversidade abrangente de características é um dos motivos pelos quais as chamadas de API são filtradas. Essa filtragem não apenas aumenta a eficiência e a qualidade dos modelos preditivos (i.e., se mantidas as ocorrências mais relevantes), mas também evita a sobrecarga do sistema ao lidar com uma grande quantidade de dados, o que poderia exigir um *hardware* mais poderoso. Assim, a ADBuilder busca equilibrar a riqueza das características extraídas com a capacidade de processamento disponível, garantindo uma análise eficaz dos aplicativos Android sem comprometer o desempenho.

Esse *dataset* é uma valiosa fonte de informações para a análise do comportamento de aplicativos Android, fornecendo uma visão detalhada e abrangente das características associadas a possíveis ameaças. Com a quantidade expressiva de características extraídas, é possível identificar padrões e realizar análises mais precisas, contribuindo para aprimorar a detecção de *malwares* e fortalecer a segurança dos dispositivos móveis.

Nesta fase do processo, foram esclarecidas as funcionalidades de cada módulo e o resultado final que cada um deles gera, contribuindo para a construção do *dataset* final. No entanto, durante a execução da ferramenta, é fundamental que o usuário possa acompanhar o progresso da ADBuilder por meio da interface do terminal. A fim de fornecer uma visão em tempo real do processamento de cada módulo, a Figura 28 ilustra os dados exibidos na tela, permitindo que o usuário tenha uma visão clara e detalhada do andamento da execução.

Figura 28 – Dados do processamento inicial da ADBuilder via terminal.

Fonte: Autor (2023).

Conforme descrito na Figura 28, no início do processamento da ADBuilder os dados são exibidos ao usuário por meio da interface do terminal. À medida que o módulo de geração processa pelo menos um APK, são mostrados os dados do *dataset* em construção, incluindo seu tamanho em bytes, a quantidade de amostras e características incluídas até o momento. Quando um módulo conclui o processamento de todos os dados, no caso dos 350 APKs, essa informação é destacada em verde, indicando que o módulo finalizou seus processos.

4.7.1 Experimentos

A subseção de experimentos foi conduzida com o objetivo de avaliar o desempenho e a eficácia da ferramenta ADBuilder na construção de conjuntos de dados para detecção de *malwares* no sistema operacional Android. Os experimentos foram divididos em dois grupos distintos, cada um focando em aspectos específicos relacionados aos tamanhos dos *datasets*.

O primeiro experimento envolveu a criação de três conjuntos de dados contendo 150, 250 e 350 APKs, respectivamente. Essa abordagem permitiu analisar o comportamento da ferramenta em relação ao aumento do número de APKs no *dataset*.

Foram coletadas métricas relevantes, como tempo de execução, utilização da CPU e consumo de memória em cada etapa do processo, a fim de compreender o impacto desses fatores no desempenho da ferramenta.

O segundo experimento concentrou-se na construção de quatro conjuntos de dados com tamanhos de APKs distintos: 1 MB, 5 MB, 10 MB e acima de 10 MB, com o maior tamanho alcançando 85 MB. Essa abordagem teve como objetivo explorar a capacidade de processamento da ADBuilder com diferentes tamanhos de arquivos APK e verificar possíveis variações no desempenho da ferramenta em função do tamanho do APK.

Ambos os experimentos foram realizados utilizando um conjunto de métricas bem definidas, possibilitando uma análise comparativa entre os diferentes cenários. As métricas obtidas representam uma média extraída dos experimentos, fornecendo informações valiosas sobre a eficiência e a escalabilidade da ADBuilder. Esses resultados são fundamentais para compreender as capacidades da ferramenta e oferecer orientações práticas para a utilização adequada no processo de construção de conjuntos de dados destinados à detecção de *malwares*.

Os experimentos foram realizados em um computador com processador Intel(R) Core(TM) i7-1185G7 3.00GHz 11th 8 *Cores*, 32GB RAM, HD, usando uma Máquina Virtual de 4 *Cores* com GNU/Linux Ubuntu 20.04.3 *LTS Desktop* 64 bit. As métricas coletadas foram o tempo médio (em segundos), uso de CPU (em porcentagem) e uso de memória RAM (em KiB).

Vale ressaltar que, quando o consumo de CPU excede 100%, isso significa que a capacidade de processamento disponível em um único núcleo ou *thread* está sendo ultrapassada. Normalmente, quando a CPU está operando em sua capacidade máxima, é relatado um uso de 100%. No entanto, em sistemas com recursos adicionais, como a capacidade de operar em vários núcleos ou *threads* simultaneamente, é possível que o consumo de CPU seja relatado acima de 100%. Por exemplo, um valor de 130% indica que a CPU está operando em sua capacidade máxima e utilizando recursos além de uma única *thread*.

Em relação ao experimento 1 (um), a ferramenta ADBuilder foi submetida a três execuções para avaliar seu desempenho na construção de conjuntos de dados para detecção de *malwares* no sistema operacional Android. Os experimentos envolveram a criação de *datasets* com 150, 250 e 350 APKs, e várias métricas foram coletadas para analisar o tempo de execução, a utilização da CPU e o consumo de memória em cada etapa do processo, como é apresentado na Tabela 11. O Experimento consistiu em avaliar

o desempenho da ferramenta em relação a variação da quantidade de entradas, ou seja, os aplicativos Android. É importante ressaltar que o módulo de *download* e extração foram executados com 5 processos em paralelo.

Tabela 11 – Resultados dos Experimentos com variação do tamanho de entradas na ferramenta

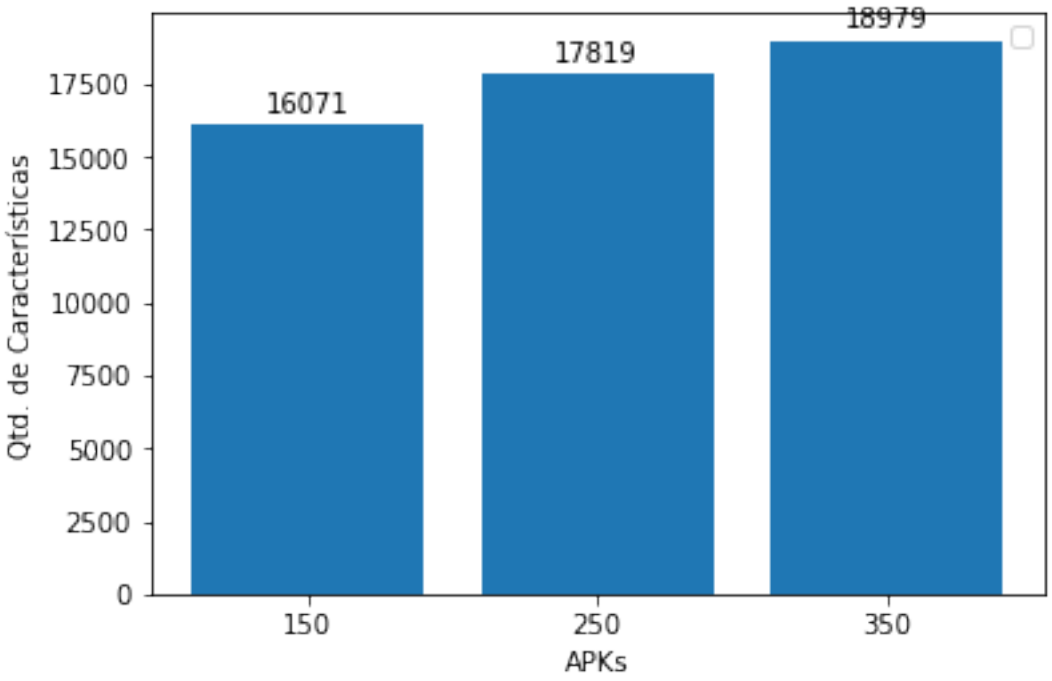
Testes (média)		**Módulos** (5 processos no *download* e extração)				
Métrica	**Qtd. de APKs**	**Download**	**Extração**	**Rotulação**	**Geração**	
					Geração	Concatenação
Tempo (s)	150	107,93	26,17	1,90	1,05	14,11
	250	114,58	30,48	1,99	2,18	24,38
	350	121,97	31,85	1,92	6,08	43,29
CPU (%)	150	0,02	116,56	48,34	122,30	100,16
	250	0,3	108,89	51,97	100,52	100,00
	350	0,02	109,23	53,05	88,52	99,86
RAM (KiB)	150	13.674,88	745.323,12	70.441,73	65.776,18	136.656,21
	250	13.742,96	749.403,17	69.615,71	65.833,48	164.292,69
	350	13.685,44	793.060,43	70.506,72	66.032,94	196.520,32

Fonte: Autor (2023)

Os resultados do primeiro experimento, conforme apresentados na Tabela 11, forneceram *insights* valiosos sobre o desempenho da ferramenta ADBuilder em relação ao tempo de execução, utilização da CPU e consumo de memória RAM. Observou-se que o tempo de execução aumentou à medida que o número de APKs no *dataset* aumentou, o que era esperado. Isso ocorre porque o processamento de um maior volume de dados requer mais tempo. Em relação à utilização da CPU, observou-se uma relativa estabilidade, embora tenham ocorrido picos em determinadas etapas do processo. Esses picos podem ser explicados pela intensidade computacional exigida em determinadas operações. Já o consumo de memória RAM mostrou variações, sem um padrão específico

identificado.

Figura 29 – Gráfico do crescimento de APKs X Características



Fonte: Autor (2023).

Conforme ilustrado na Figura 29, é apresentado o crescimento progressivo do número de características para cada um dos três *datasets* gerados pela ADBuilder no primeiro experimento. É interessante notar que, mesmo com uma diferença de apenas 100 APKs entre cada conjunto de dados, o aumento no número de características é considerável. Para o *dataset* de 150 APKs, foram registradas 16071 características, enquanto o conjunto de 250 APKs apresentou um acréscimo para 17819 características. Já o *dataset* final, composto por 350 APKs, exibiu um crescimento ainda maior, totalizando 18979 características. Esses resultados revelam um padrão consistente de expansão no número de características à medida que mais APKs são incluídos no *dataset*. Essa tendência é esperada, pois cada aplicativo adicionado contribui com novas informações e comportamentos que enriquecem a análise.

Esses resultados têm um impacto importante na utilização da ferramenta ADBuilder. Os usuários devem estar cientes de que o tempo de execução, a utilização da CPU e o consumo de memória podem aumentar consideravelmente com *datasets* maiores. É fundamental considerar a capacidade de processamento e a disponibilidade de recursos do sistema antes de executar a ferramenta. Recomenda-se monitorar o desempenho do sistema durante a execução e realizar testes preliminares com *datasets* menores para

entender melhor os requisitos de recursos da ferramenta.

Em relação ao experimento 2 (dois), são apresentados os resultados da execução da ADBuilder em quatro conjuntos distintos contendo 100 APKs de tamanhos similares (1MB, 5MB, 10MB, e maior que 10MB). A execução da ferramenta utilizou 3 processos nos módulos de **download** e **extração** de características.

Tabela 12 – Resultados dos Experimentos com variação no tamanho dos APKs.

Testes (média)		**Módulos**			
Métrica	**Tamanho**	**Download**	**Extração**	**Rotulação**	**Geração**
Tempo (s)	1 MB	8,26	4,96	1,35	3,53
	5 MB	16,32	34,96	1,13	5,28
	10 MB	44,21	83,42	1,36	10,23
	85 MB	152,76	210,31	1,12	10,17
CPU (%)	1 MB	0,47	139,55	3,81	216,39
	5 MB	0,08	112,03	2,19	217,40
	10 MB	0,02	105,68	2,94	215,52
	85 MB	0,0	106,24	2,86	211,72
RAM (KiB)	1 MB	13.710	179.918,72	13.537,36	158.314,14
	5 MB	13.683,56	585.197,80	13.492,92	173.559,96
	10 MB	13.691,72	928.076,30	13.517,30	186.452,60
	85 MB	13.712,04	1.506.036,00	13.517,20	203.215,96

Fonte: Autor (2023)

De acordo com a Tabela 12, o tamanho dos APKs influencia o tempo de *download* e extração de características. Similarmente, o consumo de memória na extração de características e geração do *dataset* também é fortemente influenciado pelo tamanho dos APKs. Na extração, o consumo de memória é praticamente exponencial de acordo com o tamanho dos APKs. Portanto, na execução da ADBuilder, é preciso observar o número de processos paralelos de extração e o tamanho dos APKs para não ultrapassar a capacidade do ambiente de execução.

Destaca-se que, nos módulos de *download* e extração, o consumo de CPU aparenta reduzir à medida que o tamanho dos APKs aumenta. No entanto, é crucial ressaltar que o valor da CPU representa o consumo médio durante um intervalo de tempo. Como esses dois módulos envolvem várias operações de entrada e saída, o consumo médio de CPU tende a ser amortizado. Por exemplo, considerando os conjuntos de APKs

de 1MB e 10MB no módulo de extração, observamos um consumo médio de CPU de 139,55% (dois cores) em 4,96 segundos e 105,68% (dois cores) em 83,42 segundos, respectivamente. Em ambos os casos, com três processos de extração concomitantes, é necessário ter no mínimo duas unidades de processamento disponíveis para acomodar a carga total sem prejudicar significativamente o desempenho do sistema. Nesse cenário, uma unidade de processamento estará utilizando toda a sua capacidade, enquanto a outra unidade será utilizada para completar a capacidade de processamento restante. No entanto, em relação ao módulo de rotulação, que utiliza o serviço do VirusTotal, há poucas diferenças entre os diferentes conjuntos de APKs. Isso significa que o tamanho dos APKs tem pouco impacto no tempo de rotulação. O serviço do VirusTotal é capaz de lidar eficientemente com diferentes tamanhos de APKs, garantindo uma análise consistente e precisa, independentemente do tamanho do arquivo. Esses resultados reforçam a eficácia da ADBuilder em processar diferentes tamanhos de APKs de forma eficiente, aproveitando os recursos de CPU disponíveis e garantindo um desempenho consistente tanto nos módulos de *download* e extração quanto no processo de rotulação.

Sem dúvidas, compreender esses números é importante na execução da ferramenta. Sem observar esses números, um usuário poderá facilmente sobrecarregar o ambiente de execução (e.g., um servidor). Se o usuário configurar um nível alto de paralelização de módulos como *download* e extração, poderá rapidamente saturar a máquina e forçar *swapping* de memória, induzindo uma condição *trashing*.

Em ambos experimentos, foram observados que o consumo médio de memória RAM durante as etapas de extração e concatenação dos *datasets* foi relativamente alto em comparação com outras etapas do processo. Essa observação pode ser explicada pelas características específicas dessas etapas. Durante a etapa de extração, a ferramenta ADBuilder executa a engenharia reversa dos APKs, desmontando-os e extraindo as características do código-fonte subjacente. Esse processo de desmontagem e análise pode demandar recursos adicionais de memória para armazenar temporariamente os dados extraídos e executar as operações necessárias. Consequentemente, é esperado um consumo de memória mais elevado nessa fase do processo.

Já na etapa de concatenação, a ADBuilder precisa adicionar sequencialmente os dados dos APKs ao *dataset* em construção. À medida que o número de APKs aumenta, o volume de dados a ser concatenado também aumenta proporcionalmente. Esse processo de concatenação de grandes quantidades de dados pode requerer uma alocação significativa de memória, resultando em um consumo mais elevado durante essa

etapa. Essas observações indicam que, ao utilizar a ADBuilder, é importante considerar a disponibilidade de memória RAM adequada para suportar as operações de extração e concatenação, especialmente em cenários com um grande número de APKs ou com APKs de tamanho significativo. Garantir que o sistema tenha recursos suficientes de memória é fundamental para evitar problemas de desempenho e garantir a conclusão bem-sucedida do processo de construção do *dataset*.

Vale ressaltar que, em relação aos testes realizados, não foi realizada uma avaliação direta do *dataset* gerado com um modelo preditivo para avaliar sua eficiência na detecção de *malwares*. Isso se deve ao fato de que o conjunto de dados possui um número limitado de amostras de APKs. Treinar e testar um modelo com essa quantidade limitada de dados poderia resultar em resultados imprecisos e enviesados. Além disso, a necessidade de dividir o conjunto de dados em partes para treinamento e teste reduziria ainda mais o número de amostras disponíveis. Portanto, a avaliação do *dataset* gerado requer a obtenção de um conjunto de dados mais robusto e representativo para obter resultados confiáveis na avaliação da eficiência do modelo preditivo.

4.7.2 Validação da ferramenta

Durante a avaliação comparativa das características extraídas, foi realizada uma comparação entre a ferramenta ADBuilder e o VirusTotal, um dos serviços mais utilizados nessa área (SHARMA; RATTAN, 2021). O VirusTotal foi escolhido como ponto de referência devido à sua ampla adoção e à sua capacidade de fornecer uma análise completa dos APKs, retornando as características do aplicativo.

Assim como a ADBuilder, o VirusTotal utiliza a ferramenta Androguard para realizar a engenharia reversa nos aplicativos e extrair suas características. Portanto, essa comparação se baseia em uma análise consistente e confiável, utilizando uma abordagem semelhante. A utilização do VirusTotal como referência para a comparação é de extrema importância, pois permite avaliar o desempenho e a qualidade das características extraídas pela ADBuilder em relação a um serviço reconhecido e amplamente utilizado.

A comparação revelou que a ADBuilder foi capaz de extrair a mesma quantidade de características que o VirusTotal, observando a Tabela 13, no entanto, a ADBuilder demonstrou uma vantagem significativa ao extrair e mapear características adicionais, como chamadas de API e *Opcodes*, que não foram mapeadas pelo VirusTotal. Esse resultado ressalta a capacidade da ADBuilder em extrair e mapear um conjunto mais

abrangente de características, o que pode ser de extrema importância na análise do comportamento de *malwares*.

Tabela 13 – Comparação de características com o VirusTotal.

Nome do APK	Características	ADBuilder	VirusTotal
Shuxue	Atividades	6	6
	Serviços	3	3
	Intenções	3	3
	Receptores	0	0
	Provedores	1	1
	Permissões	7	7
	Opcodes	44	0
	Chamadas de API	47	0
Avrasya Spor Kulup	Atividades	4	4
	Serviços	4	4
	Intenções	12	12
	Receptores	5	5
	Provedores	2	2
	Permissões	6	6
	Opcodes	194	0
	Chamadas de API	4.372	0

Fonte: Autor (2023)

Um exemplo que representa tal diferença pode ser observado no segundo APK utilizado na comparação, onde foi possível extrair um total de 4.372 chamadas de API apenas com a ADBuilder. Essa quantidade expressiva de informações permite um melhor mapeamento do comportamento de um *malware*, proporcionando uma análise mais precisa e abrangente.

Além da comparação entre a extração de características da ADBuilder e do VirusTotal, foi realizada uma análise comparativa com outra ferramenta de extração de características chamada Apktool[46]. O Apktool é uma ferramenta que realiza a engenharia reversa de APKs, permitindo a desmontagem e montagem de aplicativos Android. No experimento, o aplicativo Android utilizado foi o Qubla, e o Apktool foi empregado para desmontar o aplicativo e analisar o arquivo *AndroidManifest.xml*, que contém informações essenciais do APK, como permissões, intenções, atividades, serviços, receptores e provedores. Os resultados dessa comparação estão apresentados na Tabela 14, onde é

[46]https://ibotpeaches.github.io/Apktool/

possível observar que tanto a ADBuilder quanto o Apktool extraíram a mesma quantidade de características em todas as categorias. Esse resultado reforça a eficácia e validade da ferramenta ADBuilder, demonstrando que ela é capaz de extrair características de forma equivalente a uma ferramenta estabelecida como o Apktool, evidenciando seu potencial na análise de aplicativos Android. Os dados dessa comparação estão no repositório Github deste projeto[47].

Tabela 14 – Comparação de características com Apktool.

Nome do APK	Características	ADBuilder	Apktool
Qubla	Atividades	10	10
	Serviços	15	15
	Intenções	27	27
	Receptores	16	16
	Provedores	5	5
	Permissões	32	32

Fonte: Autor (2023)

Além da validação entre a extração de características da ADBuilder com o VirusTotal e o Apktool, foi realizada uma comparação entre conjuntos de dados. Durante a comparação entre os *datasets* gerados pela ADBuilder e o Drebin-215, um *subset* do *The Drebin Dataset* (ARP et al., 2014) amplamente utilizado na literatura para detecção de *malwares* Android, foram observadas diferenças significativas em termos de características e representatividade, que podem ser observadas na Tabela 15.

Tabela 15 – Comparação de *datasets*.

Características	ADBuilder	Drebin-215
Intenções	903	23
Permissões	422	114
Opcodes	223	0
Chamadas de API	16.265	78

Fonte: Autor (2023)

Um dos *datasets* finais da ADBuilder foi construído a partir de 250 APKs, resultando em um conjunto de dados com 17819 características extraídas, incluindo metadados. No entanto, apenas as permissões, chamadas de API, intenções e *Opcodes*

[47] https://github.com/Overycall/ADBuilder-TCC/tree/main/tests/Comparação%20com%20Apktool

foram selecionadas e incluídas no *dataset*. Vale ressaltar que as chamadas de API são filtradas para incluir apenas aquelas que são mapeadas pelo pacote Android. Essas características foram escolhidas por serem amplamente utilizadas e consideradas representativas na descrição do comportamento de aplicações maliciosas (SHARMA; RATTAN, 2021).

Em relação à comparação específica das características dos *datasets*, apresentado na Tabela 15, a ADBuilder apresentou 422 permissões, enquanto o Drebin-215 possuía apenas 114. Em termos de intenções, a ADBuilder registrou 903, em contraste com as 23 do Drebin-215. Em relação aos *Opcodes*, a ADBuilder extraiu 223, enquanto o Drebin-215 não incluiu nenhuma característica desse tipo. Por fim, em relação às Chamadas de API, a ADBuilder contou com um total de 16265, enquanto o Drebin-215 registrou apenas 78.

Essa comparação evidencia as vantagens do *dataset* gerado pela ADBuilder em relação ao Drebin-215. Mesmo com apenas 250 APKs, a ADBuilder conseguiu extrair e incluir um conjunto mais abrangente e diversificado de características do que o Drebin-215, o qual contém cerca de 15 mil amostras de APKs. Essa ampla variedade de informações permite uma análise mais detalhada e abrangente do comportamento das aplicações, o que contribui para melhorar a eficácia dos modelos de detecção de *malwares* baseados no *dataset* da ADBuilder. Vale lembrar que ao selecionar cuidadosamente as características mais relevantes e incluí-las no *dataset*, é possível aumentar a eficiência do aprendizado dos modelos preditivos, fornecendo uma visão mais completa dos padrões e características associadas a possíveis ameaças.

Em resumo, a comparação entre a ADBuilder e o VirusTotal revelou que a ADBuilder possui uma capacidade abrangente de extração de características, incluindo todas as características mapeadas pelo VirusTotal e também adicionando chamadas de API e *Opcodes* que não são mapeados pelo serviço. Essa abordagem mais completa e detalhada permite uma análise mais precisa e abrangente do comportamento de aplicações maliciosas. Ademais, ao comparar os conjuntos de dados da ADBuilder e do Drebin-215, observou-se que o *dataset* gerado pela ADBuilder é mais diversificado e representativo em termos de características. Essa diversidade pode contribuir significativamente para a melhoria da eficácia dos modelos de detecção de *malwares*. Além disso, ao realizar a comparação com o Apktool, constatou-se que a ADBuilder e o Apktool extraíram a mesma quantidade de características em todas as categorias, validando ainda mais a eficiência da ADBuilder como uma ferramenta de extração de características. Esses

resultados reforçam a importância e o potencial da ADBuilder na análise e detecção de ameaças em aplicativos Android.

4.8 Síntese da Pesquisa

A partir do referencial teórico sobre computação móvel, segurança da informação e de sistemas, sistema operacional Android, *datasets* e aprendizado de máquina, identificou-se a necessidade de desenvolver uma solução eficiente para avaliar APKs no contexto de detecção de *malwares* para o sistema operacional Android. Com base nisso, foi proposta uma solução composta por quatro módulos.

O primeiro módulo consiste no *download* dos APKs por meio do repositório AndroZoo, enquanto o segundo módulo realiza a extração das características estáticas desses APKs utilizando a ferramenta Androguard. Em seguida, o terceiro módulo utiliza a API do VirusTotal para rotular os APKs, e o quarto módulo constrói o *dataset* com base nos dados de extração e rotulação.

Aplicando princípios de engenharia de *software*, a solução foi desenvolvida em fases. Inicialmente, os dois primeiros módulos foram implementados, testados e validados, posteriormente, os módulos restantes foram construídos. Durante os testes, foram realizados dois experimentos: (a) o primeiro considerando diferentes tamanhos de entradas, i.e., construir *datasets* com 150, 250 e 350 APKs; e (b) o segundo considerando diferentes tamanhos de APKs, realizando quatro conjuntos de testes com 100 APKs cada, variando de 1 MB, 5 MB, 10 MB e acima de 10 MB até 85 MB.

Os resultados dos testes demonstraram que a solução proposta é viável, culminando na construção do *dataset* para detecção de *malwares*. Concluiu-se, portanto, que a pesquisa atingiu seu objetivo de desenvolver uma ferramenta integrada e automatizada capaz de construir conjuntos de dados atualizados para detecção de *malwares* no sistema operacional Android. As atividades realizadas, em conformidade com os objetivos específicos, incluíram a definição do problema de pesquisa, levantamento do referencial teórico, pesquisa sobre elementos a serem integrados na solução, construção da arquitetura e modelagem da ferramenta, implementação, teste e validação, análise e descrição dos resultados obtidos, documentação e escrita do TCC.

Dessa forma, os resultados indicam a viabilidade da solução proposta, que poderá ser uma contribuição significativa para a área de segurança de dispositivos móveis e detecção de *malwares* no sistema operacional Android.

5 CONSIDERAÇÕES FINAIS

No decorrer deste Trabalho de Conclusão de Curso, foi desenvolvida a ferramenta ADBuilder, uma solução integrada e automatizada para a construção de conjuntos de dados destinados à detecção de *malwares* no sistema operacional Android. Os resultados obtidos por meio dos experimentos realizados demonstraram a eficiência e a viabilidade da ferramenta na construção de conjuntos de dados atualizados, mesmo diante de variações na quantidade e no tamanho dos APKs. Ao considerar as métricas de tempo, uso de CPU e consumo de memória RAM, foi observado que o consumo médio de RAM durante a extração e a concatenação foi relativamente alto, devido à natureza do processo de engenharia reversa e à adição sequencial de dados ao *dataset* em construção. No entanto, a ferramenta mostrou-se robusta e capaz de atender às necessidades identificadas no início do trabalho. Recomenda-se aos usuários da ferramenta que estejam atentos ao consumo de recursos do sistema, especialmente em relação à memória RAM, e que mantenham a ferramenta atualizada para lidar com o dinamismo das ameaças e a evolução do sistema operacional Android.

Em suma, esta solução poderá contribuir para a área de segurança da informação ao fornecer uma solução prática e automatizada para a construção de *datasets* de detecção de *malwares*, e sugere-se que futuros estudos e aprimoramentos sejam realizados para a aplicação da ferramenta em cenários reais de detecção e prevenção. É importante ressaltar que, para utilizar a ferramenta ADBuilder, cada usuário deverá criar sua própria chave de API do AndroZoo, pois a chave utilizada nesta pesquisa é exclusiva do projeto. Além disso, a chave de API do VirusTotal será disponibilizada, porém é essencial que cada usuário tenha sua própria chave, considerando as limitações diárias impostas pelas chaves gratuitas. Essas considerações garantem a responsabilidade individual na obtenção e uso das chaves de API necessárias para o pleno funcionamento da ferramenta ADBuilder.

Salienta-se que como resultado do presente estudo, foram publicados três artigos científicos relevantes para a área de detecção de *malwares* Android. O primeiro artigo[48], intitulado "Análise do Impacto de Viés nos Conjuntos de Dados para Detecção de *Malwares* Android", teve como objetivo realizar experimentos para avaliar o desempenho de modelos preditivos treinados com conjuntos de dados defasados em relação a conjuntos de dados mais atualizados. O segundo artigo[49], uma versão estendida do primeiro, ampliou o escopo dos experimentos, investigando o impacto do treinamento

[48] https://sol.sbc.org.br/index.php/errc/article/view/18543
[49] https://revistas.setrem.com.br/index.php/reabtic/article/view/432

de modelos com dados defasados em diferentes períodos de tempo, utilizando amostras e características semelhantes. Por fim, o terceiro artigo[50] apresentou a ADBuilder como uma ferramenta inovadora para a construção automatizada de conjuntos de dados para detecção de *malwares* Android. Essas contribuições enfatizam a importância da atualização constante dos conjuntos de dados e fornecem *insights* valiosos para o desenvolvimento de técnicas de detecção de *malwares* mais eficientes e precisas.

[50]https://sol.sbc.org.br/index.php/sbseg_estendido/article/view/21703

6 TRABALHOS FUTUROS

Além das contribuições alcançadas com a implementação da ferramenta ADBuilder, existem diversas oportunidades para trabalhos futuros. Uma delas é a integração da ferramenta com serviços de armazenamento em nuvem, permitindo o envio automatizado de dados gerados durante a construção do *dataset* para um banco de dados na nuvem. Isso possibilitaria o armazenamento seguro, compartilhamento e acesso fácil aos dados por pesquisadores e profissionais da área. Outra perspectiva interessante é a expansão da ADBuilder para a extração de características dinâmicas dos APKs, além das características estáticas já contempladas. Isso permitiria uma análise mais abrangente do comportamento dos aplicativos em tempo de execução, melhorando a detecção de *malwares* e a identificação de comportamentos suspeitos. Por fim, explorar técnicas avançadas de mineração de dados e aprendizado de máquina, como algoritmos de aprendizado profundo, pode elevar ainda mais a eficiência na detecção de ameaças e contribuir para o desenvolvimento de soluções robustas de segurança para o sistema operacional Android. Esses trabalhos futuros têm o potencial de aprimorar significativamente a ADBuilder, fortalecendo sua capacidade de auxiliar na detecção e combate a *malwares* Android.

REFERÊNCIAS

ALECRIM, E. **Android 13 Tiramisu: as principais novidades da nova versão do sistema. Tecnoblog**. 2022. Disponível em: <https://tecnoblog.net/noticias/2022/05/20/android-13-tiramisu-principais-novidades-do-sistema/>. Acesso em: 15 de set. de 2022.

ALLIX. Are your training datasets yet relevant? In: SPRINGER. **ESSoS**. 2015. p. 51–67.

ALLIX, K. et al. Androzoo: Collecting millions of android apps for the research community. In: IEEE. **2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)**. 2016. p. 468–471.

ALMENARA, H. **Qual o sistema operacional de celular mais usado do mundo?** 2022. Disponível em: <https://www.terra.com.br/byte/qual-o-sistema-operacional-de-celular-mais-usado-do-mundo,76d0aa218d978b956edb0aa85695b94aye1s2sah.html#:~:text=Android%20(71%2C47%25)&text=Segundo%20o%20levantamento%20do%20site,royalties%20Ãă%20Gigante%20das%20Pesquisas.>. Acesso em: 10 de nov. de 2022.

ANDROID. **Arquitetura da plataforma**. 2020. Disponível em: <https://developer.android.com/guide/platform?hl=pt-br>.

ANDROID. **Site Oficial do Android**. 2022. Disponível em: <https://developer.android.com>. Acesso em: 05 de nov. de 2022.

ARP, D. et al. Drebin: Effective and explainable detection of android malware in your pocket. In: **Ndss**. 2014. v. 14, p. 23–26.

BAUMGÄRTNER, L. et al. Andro lyze: A distributed framework for efficient android app analysis. In: IEEE. **2015 IEEE International Conference on Mobile Services**. 2015. p. 73–80.

BICUDO, E. **Android 13: Google divulga requisitos mínimos para nova versão do sistema**. 2022. Disponível em: <https://www.tudocelular.com/google/noticias/n195408/android-13-google-requisitos-minimos-sistema.html#:~:text=Segundo%20Jason%20Bayto%20do%20programa,o%20Android%2012%20Go%20atualmente.>. Acesso em: 05 de nov. de 2022.

BINE. Estudo de segurança em dispositivos móveis. **Departamento de ciência da computação. Semana acadêmica. Universidade do centro-oeste. UNICENTRO. Guarapuava**, 2016.

BORGES, M. M. et al. Construção de um conjunto de dados para análise estática de ransomwares. In: SBC. **Anais Estendidos do XVII Simpósio Brasileiro de Sistemas de Informação**. 2021. p. 41–44.

CABRAL. Segurança em dispositivos móveis: Um estudo sobre a adoção de boas práticas para proteção em celulares. In: SBC. **Anais do XLVIII Seminário Integrado de Software e Hardware**. 2021. p. 58–68.

CATAK, F. O.; YAZI, A. F. A benchmark api call dataset for windows pe malware classification. **arXiv e-prints**, p. arXiv–1905, 2019.

COSTA. Comparativo entre gerenciadores de banco de dados para aplicação android. **Revista TIS**, v. 4, n. 1, 2016.

DELLIGATTI, L. **SysML distilled: A brief guide to the systems modeling language**. : Addison-Wesley, 2013.

DHALARIA et al. A hybrid approach for android malware detection and family classification. International Journal of Interactive Multimedia and Artificial Intelligence . . . , 2021.

DÜZGÜN, B. et al. New datasets for dynamic malware classification. **arXiv preprint arXiv:2111.15205**, 2021.

FIGUEIREDO, C. M.; NAKAMURA, E. Computação móvel: Novas oportunidades e novos desafios. **T&C Amazônia**, ano, v. 1, n. 2, p. 21, 2003.

FOWLER, M. **UML distilled: a brief guide to the standard object modeling language**. : Addison-Wesley Professional, 2004.

GHEDIN, R. **Visão geral do ART**. 2014. Disponível em: <https://manualdousuario.net/art-android-runtime/>. Acesso em: 10 de nov. de 2022.

GOMES, R. C. et al. Sistema operacional android. **Universidade Federal Fluminense**, p. 27, 2012.

HOEPERS. **Fundamentos de Segurança da Informação**. : Recuperado el junio, 2019.

HOPPEN, J. **Datasets, o que são e como utilizá-los**. 2018. Disponível em: <https://www.aquare.la/datasets-o-que-sao-e-como-utiliza-los/>. Acesso em: 18 de jan. de 2023.

HUTCHINSON. Are we really protected? an investigation into the play protect service. In: IEEE. **2019 IEEE International Conference on Big Data (Big Data)**. 2019. p. 4997–5004.

INSIDE, R. **Detecções de malware bancário no Android aumentaram 428% no último ano. TI Inside**. 2022. Disponível em: <https://tiinside.com.br/17/02/2022/deteccoes-de-malware-bancario-no-android-aumentaram-428-no-ultimo-ano/>. Acesso em: 25 de set. de 2022.

ITO, G. C.; FERREIRA, M.; SANT'ANA, N. Computação móvel: Aspectos de gerenciamento de dados. **INPE-Instituto Nacional de Pesquisas espaciais**, v. 10, p. 17–18, 2003.

JACOBSON, I.; NG, P.-W. **Aspect-oriented software development with use cases (addison-wesley object technology series)**. : Addison-Wesley Professional, 2004.

JAGIELSKI, M. et al. Manipulating machine learning: Poisoning attacks and countermeasures for regression learning. In: IEEE. **2018 IEEE Symposium on Security and Privacy (SP)**. 2018. p. 19–35.

JAIN. Overview and importance of data quality for machine learning tasks. In: **Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining**. 2020. p. 3561–3562.

JULIANA. **Gerenciamento de Memória – Swapping**. 2017. Disponível em: <https://jkolb.com.br/gerenciamento-de-memoria-swapping/>. Acesso em: 10 de nov. de 2022.

KLIMA, V. Finding md5 collisions–a toy for a notebook. **Cryptology ePrint Archive**, 2005.

KLIMA, V. Tunnels in hash functions: Md5 collisions within a minute. **Cryptology ePrint Archive**, 2006.

KUMAR, U. S.; YADAV, A.; SINGH, V. Detecting malware in android applications by using androguard tool and xgboost algorithm. In: IEEE. **2022 IEEE 9th Uttar Pradesh Section International Conference on Electrical, Electronics and Computer Engineering (UPCON)**. 2022. p. 1–6.

LASHKARI, A. H. et al. Toward developing a systematic approach to generate benchmark android malware datasets and classification. In: IEEE. **ICCST**. 2018. p. 1–7.

LI, L. et al. Androzoo++: Collecting millions of android apps and their metadata for the research community. **arXiv preprint arXiv:1709.05281**, 2017.

LIMA, C. S. P.; BARBOSA, S. d. F. F. Aplicativos móveis em saúde: caracterização da produção científica da enfermagem brasileira. **Revista eletrônica de enfermagem**, v. 21, 2019.

LIMA, W. D. Android e a influência do sistema operacional linux. **Tecnologias em Projeção**, v. 8, n. 1, p. 100–111, 2017.

LIU, K. et al. A review of android malware detection approaches based on machine learning. **IEEE Access**, IEEE, v. 8, p. 124579–124607, 2020.

LOCKHEIMER, H. **O Android é para todos**. 2022. Disponível em: <https://www.android.com/intl/pt-BR_br/everyone/#:~:text=O%20Google%20anuncia%20mais%20de,400%20fabricantes%20e%20500%20operadoras.>

MAHINDRU et al. Somdroid: Android malware detection by artificial neural network trained using unsupervised learning. **Evolutionary Intelligence**, Springer, 2020.

MAHINDRU et al. Mldroid—framework for android malware detection using machine learning techniques. **Neural Computing and Applications**, Springer, v. 33, n. 10, p. 5183–5240, 2021.

MARQUES, A. **Android 12: confira todos os recursos do novo sistema. Tecnoblog**. 2021. Disponível em: <https://tecnoblog.net/especiais/android-12-os-novos-recursos-e-os-celulares-com-atualizacao-garantida/>. Acesso em: 15 de set. de 2022.

MARTÍN, A.; LARA-CABRERA, R.; CAMACHO, D. A new tool for static and dynamic android malware analysis. In: WORLD SCIENTIFIC. **Data Science and Knowledge Engineering for Sensing Decision Support: Proceedings of the 13th International FLINS Conference (FLINS 2018)**. 2018. p. 509–516.

MATEUS, G. R.; LOUREIRO, A. A. F. Introdução à computação móvel. DCC/IM, COPPE/UFRJ, 1998.

MAZIERO, C. A. Sistemas operacionais: conceitos e mecanismos. **Livro aberto**, 2014.

MEYER, M. **A história do Android. Oficina da Net**. 2020. Disponível em: <https://www.oficinadanet.com.br/post/13939-a-historia-do-android>. Acesso em: 14 de set. de 2022.

MODERNOS, E. Segurança em sistemas. 2015.

OLIVEIRA, A. de et al. Um estudo sobre o sistema operacional android. **REVISTA DE TRABALHOS ACADÊMICOS-CAMPUS NITERÓI**, n. 1, 2014.

PAN, C.-T. et al. Vein pattern locating technology for cannulation: a review of the low-cost vein finder prototypes utilizing near infrared (nir) light to improve peripheral subcutaneous vein selection for phlebotomy. **Sensors**, MDPI, v. 19, n. 16, p. 3573, 2019.

PAN, Y. et al. A systematic literature review of android malware detection using static analysis. **IEEE Access**, IEEE, v. 8, p. 116363–116379, 2020.

PONTES, J. et al. Ferramentas de extração de características para análise estática de aplicativos android. In: SBC. **Anais da XIX Escola Regional de Redes de Computadores**. 2021. p. 37–42.

PRESSMAN, R. S.; MAXIM, B. R. **Engenharia de software-9**. : McGraw Hill Brasil, 2021.

RODRIGUES, R. **Ataques a dispositivos móveis crescem 124% em março. Kaspersky**. 2020. Disponível em: <https://www.kaspersky.com.br/blog/phishing-covid-smartphone-pesquisa/14663/>. Acesso em: 18 de out. de 2022.

ROMÃO. Lgpd: Descomplicado. In: **Congresso de Tecnologia-Fatec Mococa**. 2021. v. 5, n. 2.

ROTONDO, G. Prosec-uma solução para o controle de processos maliciosos na plataforma android. Universidade Federal do Pampa, 2016.

SAHAY. Evolution of malware and its detection techniques. In: **Information and Communication Technology for Sustainable Development**. : Springer, 2020. p. 139–150.

SANTIAGO, A. **Primeira multa aplicada da LGPD**. 2023. Disponível em: <https://www.uol.com.br/tilt/noticias/redacao/2023/07/10/primeira-multa-aplicada-lgpd-brasil.htm>. Acesso em: 17 de jul. de 2023.

SANTOS dos. Implementação de um simulador de algoritmos de escalonamento de processos. 2009.

SCHENDES, W. **36 aplicativos maliciosos são identificados no Android; Confira lista. Olhar Digital**. 2022. Disponível em: <https://olhardigital.com.br/2022/07/27/seguranca/aplicativos-maliciosos-android/>. Acesso em: 13 de set. de 2022.

SHARMA, T.; RATTAN, D. Malicious application detection in android—a systematic literature review. **Computer Science Review**, Elsevier, v. 40, p. 100373, 2021.

SILVA, E. L. D.; MENEZES, E. M. Metodologia da pesquisa e elaboração de dissertação. **UFSC, Florianópolis, 4a. edição**, v. 123, 2005.

SILVEIRA, D. T.; CÓRDOVA, F. P. A pesquisa científica. **Métodos de pesquisa. Porto Alegre: Editora da UFRGS, 2009. p. 33-44**, 2009.

SINGH, R. An overview of android operating system and its security. **int. journal of Engineering Research and Applications**, v. 4, n. 2, p. 519–521, 2014.

SOARES, L. F. G.; GUIDO, L.; COLCHER, S. Redes de computadores: das lans, mans e wans às redes atm. Campus Rio de Janeiro, 1995.

SUGUNAN, K.; KUMAR, T. G.; DHANYA, K. Static and dynamic analysis for android malware detection. In: **Advances in Big Data and Cloud Computing**. : Springer, 2018. p. 147–155.

TAM. The evolution of android malware and android analysis techniques. **ACM Computing Surveys (CSUR)**, ACM New York, NY, USA, v. 49, n. 4, p. 1–41, 2017.

TEAM, B. et al. Sanddroid: An apk analysis sandbox. **Xi'an Jiaotong University: Xi'an, China**, 2014.

VENTURA, F. **Android 11 chega à versão final; veja o que muda na atualização. Tecnoblog**. 2020. Disponível em: <https://tecnoblog.net/noticias/2020/09/08/android-11-chega-a-versao-final-veja-o-que-muda-na-atualizacao/>. Acesso em: 15 de set. de 2022.

VILANOVA, L. et al. Adbuilder: uma ferramenta de construção de datasets para detecção de malwares android. In: **Anais Estendidos do XXII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais**. Porto Alegre, RS, Brasil: SBC, 2022. p. 143–150. ISSN 0000-0000. Disponível em: https://sol.sbc.org.br/index.php/sbseg_estendido/article/view/21703.

VILANOVA, L. et al. Análise do impacto de viés nos conjuntos de dados para detecção de malwares android. In: SBC. **Anais da XIX Escola Regional de Redes de Computadores**. 2021. p. 61–66.

VILELA, G. Lgpd: Um estudo sobre as principais responsabilidades e penalidades previstas na lei. Pontifícia Universidade Católica de Goiás, 2021.

WANG, H. et al. Rmvdroid: towards a reliable android malware dataset with app metadata. In: IEEE. **IEEE/ACM 16th MSR**. 2019. p. 404–408.

WANG, W. et al. Constructing features for detecting android malicious applications: issues, taxonomy and directions. **IEEE access**, IEEE, v. 7, p. 67602–67631, 2019.

YU, A.; AGARWAL, Y.; HONG, J. I. Analysis of longitudinal changes in privacy behavior of android applications. **arXiv preprint arXiv:2112.14205**, 2021.

ZHANG, X.; BREITINGER, F.; BAGGILI, I. Rapid android parser for investigating dex files (rapid). **Digital Investigation**, Elsevier, v. 17, p. 28–39, 2016.