ADBuilder: uma Ferramenta de Construção de Datasets para Detecção de Malwares Android

Lucas Vilanova[1], Diego Kreutz[1], Joner Assolin[2], Vagner Quincozes[1]
Charles Miers[3], Rodrigo Mansilha[1], Eduardo Feitosa[2]

[1]Universidade Federal do Pampa (UNIPAMPA)
{DiegoKreutz,Mansilha,{LucasVilanova,VagnerQuincozes}.aluno}@unipampa.edu.br

[2]Universidade Federal do Amazonas (UFAM)
{joner.assolin,efeitosa}@icomp.ufam.br

[3]Universidade do Estado de Santa Catarina (UDESC)
charles.miers@udesc.br

Resumo. *A maioria dos datasets existentes possuem um número limitado ou defasado de amostras, o que compromete o treinamento de modelos para detecção de malwares Android. Na literatura existem alguns trabalhos que propõem a construção de datasets, porém, nenhum deles suficientemente definido e integrado a ponto de entregar de forma automatizada e sistematizada um dataset ao usuário. Neste trabalho, propomos a ferramenta ADBuilder, que permite a construção integrada, automatizada e sistematizada de datasets atualizados para o domínio de detecção de malwares Android. A ADBuilder é constituída de quatro módulos independentes, sendo cada módulo composto por um conjunto de ferramentas ou serviços necessários para a construção de datasets atualizados.*

1. Introdução

Os *datasets* de qualidade são necessários para o treinamento e a validação de modelos preditivos eficazes para a detecção de *malwares* Android. É importante destacar que tanto a quantidade de amostras e características quanto a qualidade e atualidade dos dados impactam as métricas resultantes dos modelos criados [Allix et al., 2015]. Um problema recorrente, que demonstramos recentemente através de duas análises abrangentes [Soares et al., 2021a, Soares et al., 2021b], é o fato da maioria dos *datasets* existentes, e amplamente utilizados para o treinamento e a validação de modelos de classificação, serem limitados em termos de quantidade de características e, principalmente, completude e atualidade dos dados. Além dos nossos estudos, outros trabalhos também detectam problemas em *datasets* específicos, e.g., problemas de marcação das amostras nos *datasets* MalGenome, Drebin, Piggybacking e AMD [Wang et al., 2019].

Em um levantamento bibliográfico que realizamos, constatamos que a maioria dos trabalhos de construção de *datasets* para a detecção de *malwares* Android carece de: etapas cruciais (e.g., consolidação de características e sanitização do *dataset*); detalhamento necessário para sua reprodutibilidade (e.g., informações sobre ferramentas de extração; informações detalhadas sobre a extração das características e a rotulação; disponibilização de código ou ferramental; *threshold* utilizado para rotular um *malware*); e um sistema integrado e automatizado para a construção de

conjuntos de dados atualizados. Por exemplo, trabalhos como [Lashkari et al., 2018, Wang et al., 2019, Mahindru and Sangal, 2020, Düzgün et al., 2021] não disponibilizam código, não detalham as ferramentas utilizadas e não incorporam etapas de consolidação de características ou sanitização do *dataset*. Essas limitações podem causar problemas como conjuntos de dados com amostras duplicadas, número reduzido de características, valores faltantes e erros de nomenclatura (e.g., mesma característica com múltiplos nomes), para citar alguns exemplos.

Na etapa da extração de características, por exemplo, observamos limitações específicas. É notável que nenhum dos trabalhos relacionados, voltados para análise estática, empregou a ferramenta AndroGuard, que pode ser considerada atualmente a mais completa e eficaz para a extração de características estáticas [Pontes et al., 2021]. Mesmo os trabalhos que consideram apenas características dinâmicas, como [Lashkari et al., 2018, Catak and Yazı, 2019], falham na discussão e análise qualitativa das diferentes ferramentas disponíveis para a extração das características. Como recomendado na literatura (e.g., [Pan et al., 2020, Pontes et al., 2021]), é fundamental realizar uma avaliação qualitativa adequada das ferramentas de extração de características e utilizar as mais adequadas para cada caso de uso.

Neste trabalho apresentamos a ferramenta ADBuilder, composta por quatro módulos independentes que, integrados, permitem a construção sistematizada e automatizada de *datasets* atualizados para o treinamento de modelos de *aprendizado de máquina* para detecção de *malwares* Android. Os quatro módulos implementam seis etapas bem definidas: (i) catalogação e (ii) rotulação de aplicativos, (iii) extração e (iv) consolidação de características, (v) criação e (vi) limpeza do *dataset*.

O restante deste trabalho está organizado como segue. Nas Seções 2 e 3 apresentamos a arquitetura e implementação da ADBuilder. Na Seção 4 descrevemos a metodologia e analisamos os resultados de uma avaliação. Por fim, na Seção 5 apresentamos as informações básicas sobre a demonstração e as considerações.

2. Arquitetura

A arquitetura da ferramenta ADBuilder (Figura 1), é composta por quatro módulos principais: (i) **download** de APKs, (ii) **extração** de características, (iii) **rotulação** das amostras e (iv) **geração** do *dataset*. A partir de uma lista de resumos criptográficos (e.g., SHA256) dos aplicativos (**entrada** da ferramenta), que pode ser obtida em repositórios de APKs como o AndroZoo[1], a ADBuilder inicia a execução dos diferentes módulos até gerar o *dataset* (**saída** da ferramenta).

O primeiro módulo recebe como entrada uma lista de resumos criptográficos dos aplicativos e realiza o **download** de cada um. Os aplicativos podem ser obtidos de repositórios como o AndroZoo ou lojas de aplicativos como Google Play, Anzhi, AppChina, 1mobile, FreewareLovers, HiApk e F-Droid. Os aplicativos cujo *download* é concluído com sucesso são adicionados à fila de extração de características. Caso não seja possível concluir o *download*, o aplicativo é descartado na etapa de extração, i.e., não será utilizado como amostra na construção do *dataset*.

[1] `https://androzoo.uni.lu`

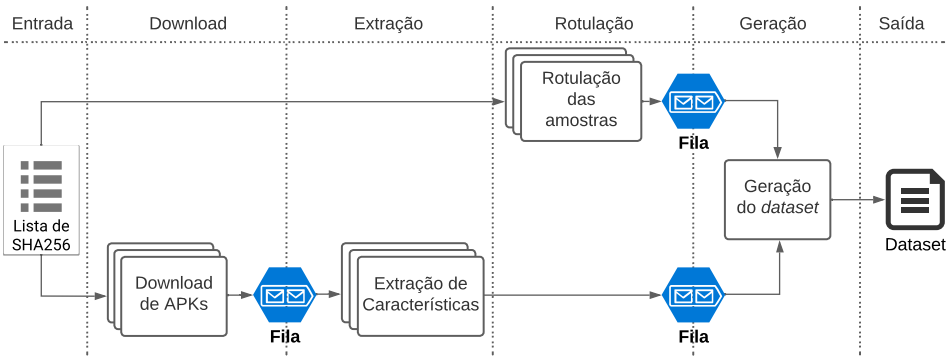

Figura 1. Arquitetura da ferramenta ADBuilder

O segundo módulo é de **extração**. Este consome os APKs da fila de *download* e extrai as características de cada um deles utilizando ferramentas específicas (e.g., AndroGuard para características estáticas). A extração é o módulo que mais consome recursos computacionais, pois envolve processos como o desempacotamento do APK e a geração de código intermediário para análise e extração (e.g., geração de grafo de chamadas de API). O resultado desse módulo é um arquivo de dados que detalha as características do aplicativo. O arquivo de características é enviado para a fila de dados de **construção** do *dataset*.

O terceiro módulo é de **rotulação**. Este utiliza ferramentas (e.g., antivirus offline) ou serviços *online* (e.g., VirusTotal) para gerar dados que permitirão aos usuários dos *datasets* rotular, no caso como benigno ou maligno, cada uma das amostras. Cada serviço pode conter um ou mais *scanners*. Por exemplo, o VirusTotal é um serviço *online* que disponibiliza, em julho de 2022, mais de 60 *scanners* para classificar um APK em benigno ou maligno. Cada *scanner* do serviço retorna um resultado para o APK. Como os resultados dos *scanners* podem divergir entre si, é necessário um método de decisão (e.g., maioria, pelo menos um voto). Visando fornecer flexibilidade, nós implementamos um sistema de contagem e limiar inferior. Assim, cabe ao usuário definir um limiar de rotulação (e.g., mais de 20 *scanners* positivos definem um aplicativo como maligno). Os dados de saída do módulo de rotulação entram em uma fila que alimenta o módulo seguinte.

O quarto e último módulo é de **geração**. Este combina as saídas das filas dos módulos de **extração** e **rotulação** para gerar o *dataset*. O trabalho principal desse módulo é realizar a junção dos dados de extração com os dados de rotulação para gerar o *dataset* com as amostras previstas na **entrada** da ADBuilder. Adicionalmente, o módulo de geração realiza também processos básicos de sanitização do *dataset*, necessários para garantir uma qualidade mínima dos dados.

É importante destacar que os módulos são temporalmente fracamente acoplados entre si, pois a conexão entre estes ocorre através de filas de dados seguindo um modelo produtor/consumidor. Dessa forma, para cada fila há um módulo que alimenta a fila com novos dados (e.g., **download**) e outro módulo que consome dados da fila ao processá-los (e.g., **geração**). Vale ressaltar também que os módulos **extração** e **rotulação** atuam tanto como produtor como consumidor, dependendo da fila. As filas diminuem o acoplamento temporal, pois os processos consumidor/-

produtor podem trabalhar com vazões distintas sem que o processo relativamente mais lento acoplado à fila atrapalhe o processo mais rápido. Além disso, os três primeiros módulos podem ser executados por múltiplas instâncias em paralelo. Esse grau de paralelismo pode acelerar a construção do *dataset*, superando limitações impostas por processos gargalos. Os serviços de rotulação, com o VirusTotal, possuem limitações por chave de API, i.e., podem ser utilizadas diversas chaves em paralelo para aumentar a vazão da rotulação, entre outros.

3. Implementação

A Figura 2 representa uma instância implementada da arquitetura (Seção 2). As filas entre os módulos foram implementadas utilizando um sistema de arquivos, arquivos de sincronização (i.e., lock) e mecanismos de controle do sistema operacional. Por exemplo, cada processo do módulo de **download** adiciona o arquivo *.apk* e um arquivo de controle em um determinado diretório. O módulo de **extração** consumirá o APK somente quando o arquivo de controle estiver liberado.

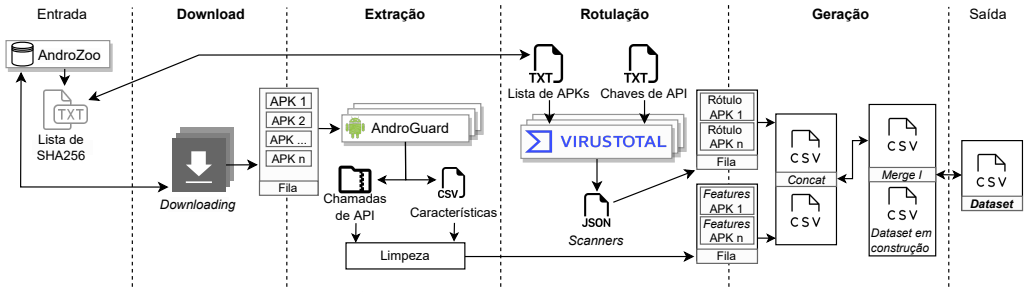

Figura 2. Implementação da ADBuilder.

Como entrada, o sistema recebe uma lista de resumos criptográficos (SHA256) dos APKs. Essa lista é obtida do AndroZoo, que contém atualmente um catálogo histórico de mais de 19 milhões de aplicativos de diferentes fontes, como Google Play, Anzhi, e AppChina. A partir dessa lista, o usuário da ADBuilder pode selecionar os APKs que desejar utilizar como amostras do *dataset*. É importante ressaltar que a lista de APKs do AndroZoo contém não apenas os metadados básicos (e.g., nome e tamanho do aplicativo), mas também metadados de rotulação (e.g., número de detecções do VirusTotal). Entretanto, a análise do VirusTotal é realizada apenas quando o aplicativo é incluído no repositório e, portanto, os metadados de rotulação podem estar defasados no momento da utilização.

Para a geração de um *dataset* atualizado e confiável, é necessário realizar uma rotulagem atualizada (nova) dos APKs. Por exemplo, os APKs *Smart Gallery 1.1* e *Smart Cleaner* foram analisados em 2019 e 2020, respectivamente. Na época, ambos os APKs foram classificados como benignos pelo AndroZoo. Entretanto, ao submetermos os APKs novamente à análise do VirusTotal, em julho de 2022, obtivemos um resultado diferente: 20 e 19 *scanners* positivos, respectivamente. É importante destacar que caberá ao usuário definir o limiar (e.g., 15 *scanners*) para classificar um APK como benigno ou maligno.

A lista de aplicativos selecionados (lista de SHA256) alimenta o processo de **download**. Esse processo é realizado através do comando `curl` parametri-

zado com uma chave de API[2] fornecida pelos mantenedores do AndroZoo. Qualquer usuário pode solicitar uma chave de API do AndroZoo. Para paralelizar o processo de download, é preciso observar o limite imposto pelo AndroZoo (atualmente, 40 *downloads* paralelos para residentes fora da Europa) e ativar o parâmetro `--download lista_de_sha256.txt -npd 40` da ferramenta adequadamente. Baseado nesses parâmetros de usuário, a lista de entrada será dividida entre os N processos de download definidos pelo usuário.

O módulo de **extração** opera com base na respectiva fila de entrada. Quando há algum APK liberado na fila, o módulo executa a extração de características e gera um arquivo CSV do aplicativo. Para evitar problemas de concorrência entre as instâncias do módulo, antes de iniciar a extração, é criado um arquivo de *lock* para sinalizar aos outros processos que o APK em questão já está no estágio de extração das características. Atualmente, para a extração de características estáticas dos APKs é utilizada a biblioteca AndroGuard[3]. Ao final da extração, um arquivo CSV contendo os metadados (e.g., SHA256, nome, pacote, versões de API) e as características (permissões, intenções, chamadas de API, atividades, serviços, provedores, receptores, *opcodes*) do APK é adicionado à fila de geração do *dataset*.

A implementação atual do módulo de **rotulação** utiliza o serviço do VirusTotal [4], que conta com mais de 60 *scanners*, cujos dados podem ser utilizados na rotulação dos aplicativos. A principal limitação do módulo está associada às chaves de API do VirusTotal. Atualmente, na versão gratuita, cada chave de API é limitada a 500 requisições por dia, 4 requisições por minuto e 15.000 requisições por mês. Se o APK já foi analisado recentemente pelo serviço, será necessária apenas uma requisição. Entretanto, caso seja um aplicativo novo, ainda não analisado pelo VirusTotal, serão necessárias duas requisições, uma para registrar a demanda e outra para coletar os resultados. Essas regras implicam que, para cada chave de API, poderão ser realizadas no máximo 250 análises de APKs por dia.

O módulo de rotulação recebe dois parâmetros de entrada: a lista de SHA256 (identificadores dos APKs) e uma lista de chaves de API do VirusTotal. A lista de SHA256 é dividida igualmente entre as chaves de API, criando filas de processamento. Cada fila é gerenciada por um processo independente, que utiliza uma chave de API específica para enviar solicitações de análise de APKs para o serviço online do VirusTotal. Cada processo pode executar uma requisição a cada 15 (ou mais) segundos, de forma a evitar o limite de 4 requisições por minuto. O retorno da análise do APK é um arquivo JSON que contém metadados gerais do aplicativo (e.g., nome, pacote, características) e os resultados de análise de cada *scanner* disponível no VirusTotal, informando se o APK foi classificado como benigno ou maligno. Ao final, é gerado um arquivo CSV, na fila compartilhada entre os módulos de rotulação e geração, contendo o resultado sumarizado da análise dos *scanners*.

Por fim, o módulo de **geração** entra em ação quando há arquivos CSV de um APK, disponibilizados pelos módulos de **extração** e **rotulação**. Primeiramente, o gerador concatena os dados de extração (características e outros metadados) e

[2] `https://androzoo.uni.lu/api_doc`

[3] `https://androguard.readthedocs.io`

[4] `https://www.virustotal.com`

rotulação (metadados dos *scanners* do VirusTotal). Em seguida, os dados passam por etapas de sanitização (e.g., transformação do formato dos dados para números inteiros e preenchimento de valores faltantes). Finalmente, os dados seguem para a construção do *dataset* propriamente dito. A construção da matriz de características do *dataset* consiste em identificar a existência (1) ou não (0), para a amostra derivada de um APK específico, de cada uma das características já contidas na matriz do *dataset*. Por exemplo, assumindo que o APK A contém 100 características e o *dataset* atual, em construção, já contém 1.000 características, no mínimo 900 características serão marcadas como não presentes para o APK A. Esse número de características pode aumentar continuamente até a finalização da construção do *dataset*.

O processo de construção do *dataset* será finalizado quando o módulo de geração receber um sinal de finalização na fila de arquivos CSV gerados pelo módulo de extração. Em outras palavras, o módulo de extração irá sinalizar o módulo de geração que não há mais amostras para serem incluídas no *dataset* em construção.

A Tabela 1 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 *scripting* 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.

Tabela 1. 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	Instalado
Módulos Python	Lxml	4.5.0
	Pandas	1.3.5
	Networkx	2.2
	Numpy	1.22.3
Sistema Operacional	Ubuntu	20.04 LTS

A ferramenta ADBuilder foi testada e utilizada em diferentes ambientes. Por exemplo, foram realizados testes em distribuições GNU/Linux Ubuntu 20.04 e 22.04. Os detalhes completos dos ambientes estão disponíveis no repositório do GitHub[5].

4. Avaliação

Na Tabela 2 apresentamos 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. As métricas coletadas foram o tempo médio (em segundos), uso de CPU (em porcentagem) e uso de memória RAM (em KiB). Os testes foram realizado em um computador com processador Intel(R)

[5]`https://github.com/Malware-Hunter/sf22-adbuilder`

Core(TM) i7-1185G7 3.00GHz 11th 8 Cores, 32GB RAM, HD, usando uma VM de 4 Cores com GNU/Linux Ubuntu 20.04.3 LTS Desktop 64 bit.

Tabela 2. Resultados dos Experimentos.

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

Como podemos observar, 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.

É interessante observar que o consumo de CPU, nos módulos de download e extração, aparenta reduzir com o aumento do tamanho dos APKs. Entretanto, é importante ressaltar que o valor da CPU representa o consumo médio pelo intervalo de tempo, ou seja, como há várias operações de entrada e saída nesses dois módulos, o consumo médio de CPU tende a ser amortizado. Tomando como exemplo os conjuntos de APKs 1MB e 10MB, para o módulo de extração, temos um consumo médio de CPU de 139,55% (dois cores) em 4,98s e 105,68% (dois cores) em 83,42s, respectivamente. Em ambos os casos, com três processos concomitantes de extração, são demandadas pelo menos duas unidades de processamento para comportar a carga total sem deixar o sistema mais lento. Já com relação ao módulo de rotulação, que utiliza o serviço do VirusTotal, há poucas diferenças entre os diferentes conjuntos de APKs, i.e., o tempo de rotulação é pouco influenciado pelos tamanhos dos APKs.

Para finalizar, é importante ressaltarmos que 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, extração e rotulação, poderá rapidamente saturar a máquina e forçar *swapping* de memória, induzindo uma condição *trashing*.

5. Considerações Finais

Demonstração. A demonstração da ferramenta será realizada através de um ambiente hospedado em um dispositivo próprio dos autores. As funcionalidades da

ferramenta serão apresentadas através dos seguintes passos: (i) apresentação dos parâmetros e opções de execução; (ii) demonstração da execução individual dos módulos; (iii) demonstração da execução dos módulos para a construção de um *dataset*; e (iv) apresentação do *dataset* gerado.

Conclusão. Apresentamos a ADBuilder, uma ferramenta constituída por quatro módulos temporalmente fracamente acoplados que permitem a automatização e sistematização da construção de *datasets* atualizados para a detecção de *malwares* em Android. A implementação da ferramenta utiliza filas para gerenciar a execução dos processos dos módulos de forma paralela. A versão atual da ferramenta permite gerar *datasets* de maneira automática. Os resultados dos testes experimentais demonstram que a ADBuilder consegue construir um *dataset* contendo 100 amostras e mais de 4.504 características em menos de 70 minutos. Acreditamos que a ferramenta é um importante passo na direção de modelos de aprendizado de máquina mais eficazes e realistas, i.e., treinados e otimizados com dados atualizados.

Agradecimentos. Esta pesquisa foi parcialmente financiada, conforme previsto nos Arts. 21 e 22 do decreto no. 10.521/2020, nos termos da Lei Federal no. 8.387/1991, através do convênio no. 003/2021, firmado entre ICOMP/UFAM, Flextronics da Amazônia Ltda e Motorola Mobility Comércio de Produtos Eletrônicos Ltda. O presente trabalho foi realizado também com apoio da CAPES – Código de Financiamento 001.

Referências

Allix, K., Bissyandé, T. F., Klein, J., and Le Traon, Y. (2015). Are your training datasets yet relevant? In *ESSoS*, pages 51–67. Springer.

Catak, F. and Yazı, A. (2019). A benchmark api call dataset for windows pe malware classification. *arXiv e-prints*, pages arXiv–1905.

Düzgün, B., Çayır, A., Demirkıran, F., Kayha, C. N., Gençaydın, B., and Dağ, H. (2021). New datasets for dynamic malware classification. *arXiv preprint arXiv:2111.15205*.

Lashkari, A. H., Kadir, A. F. A., Taheri, L., and Ghorbani, A. A. (2018). Toward developing a systematic approach to generate benchmark android malware datasets and classification. In *ICCST*, pages 1–7. IEEE.

Mahindru, A. and Sangal, A. (2020). Somdroid: Android malware detection by artificial neural network trained using unsupervised learning. *Evolutionary Intelligence*.

Pan, Y., Ge, X., Fang, C., and Fan, Y. (2020). A systematic literature review of android malware detection using static analysis. *IEEE Access*, 8:116363–116379.

Pontes, J., Costa, E., Rocha, V., Neves, N., Feitosa, E., Assolin, J., and Kreutz, D. (2021). Ferramentas de extração de características para análise estática de aplicativos android. In *WRSeg21*.

Soares, T., Mello, J., Barcellos, L., Sayyed, R., Siqueira, G., Casola, K., Costa, E., Gustavo, N., Feitosa, E., and Kreutz, D. (2021a). Detecção de Malwares Android: Levantamento empírico da disponibilidade e da atualização das fontes de dados. In *WRSeg21*.

Soares, T., Siqueira, G., Barcellos, L., Sayyed, R., Vargas, L., Rodrigues, G., Assolin, J., Pontes, J., Feitosa, E., and Kreutz, D. (2021b). Detecção de Malwares Android: datasets e reprodutibilidade. In *WRSeg21*.

Wang, H., Si, J., Li, H., and Guo, Y. (2019). Rmvdroid: towards a reliable android malware dataset with app metadata. In *IEEE/ACM 16th MSR*, pages 404–408. IEEE.