APKAnalyzer: Ferramenta de Classificação de *Malwares Android* Baseada em *Multi-view* e Seleção de Características Multiobjetivo

Philipe Fransozi[1], **Jhonatan Geremias**[1], **Eduardo K. Viegas**[1], **Altair O. Santin**[1]

[1]Programa de Pós-graduação em Informática (PPGIa)
Pontifícia Universidade Católica do Paraná (PUCPR)
Caixa Postal 17.315 – 80.242-980 – Curitiba – PR – Brasil

{philipe.hfransozi, jgeremias, eduardo.viegas, santin}@ppgia.pucpr.br

Abstract. *With the widespread use of the Android operating system, developing new techniques to address the increasing number of malicious applications for this platform has become a significant challenge. This article proposes an Android malware classification tool called APKAnalyzer, which employs three machine learning models for classification. The application's behavioral feature vector is refined using multi-view techniques, made feasible through multi-objective feature selection. This ensures that only features that improve accuracy and reduce inference time are used in model training.*

Resumo. *Com a popularização do sistema operacional Android, tornou-se um desafio desenvolver novas técnicas para enfrentar o crescente número de aplicações maliciosas para essa plataforma. Este artigo propõe uma ferramenta de classificação de malwares para Android, chamada APKAnalyzer, que tem como motor de classificação três modelos de classificação de aprendizagem de máquina. A construção do vetor de características comportamentais do aplicativo a ser analisado é aprimorado com técnicas de multi-view, cuja viabilização é possível com o uso de seleção de características multiobjetivo à medida que apenas características que melhorem a acurácia e reduzam o tempo de inferência serão utilizadas no treinamento dos modelos.*

1. Introdução

O sistema operacional *Android* estabeleceu-se como o sistema operacional mais utilizado no mundo, totalizando cerca de 3 bilhões de usuários ativos, quase três quartos do atual mercado [AndroidStats]. Ao passo que se tornou líder de mercado, o número de aplicações maliciosas para *Android*, conhecidas como malware, também aumentou ao longo do tempo. De acordo com um relatório de cibersegurança, [Kaspersky], apenas em 2023, o número de amostras de *malware* para *Android* cresceu 52%.

Em vista desse cenário, pesquisadores e a indústria buscam por novas técnicas que possam enfrentar o crescente número de novas aplicações maliciosas com ferramentas para análise e classificação dos aplicativos *Android*, conhecidos como APK. Geralmente, essas ferramentas adotam técnicas de análise estática, examinando os arquivos relacionados ao aplicativo *Android*. Dentre esses arquivos, destacam-se o arquivo *manifest.xml*, no qual estão as permissões requisitadas pela aplicação, e o arquivo *.dex*, que representa o código binário, desde onde se obtém os *opcodes* e as chamadas de API

[Darwaish and Nait-Abdesselam 2020, dos Santos et al. 2021]. Trata-se de uma abordagem amplamente adotada pela literatura em virtude de sua fácil condução e resultados promissores.

Ademais, em consonância com a crescente evolução e adoção de técnicas de inteligência artificial, a literatura apresenta abordagens [Geremias et al. 2022, dos Santos et al. 2023] para a tarefa de classificação que recorrem a técnicas baseadas em aprendizado de máquina. Nesses casos, gera-se um vetor de características comportamentais do arquivo APK, extraídas de um processo de análise estática, e o utiliza como entrada em um modelo de aprendizado de máquina [Horchulhack et al. 2024]. O modelo, por sua vez, classificará o arquivo APK como *goodware* ou *malware* [Geremias et al. 2023]. Apesar de promissores resultados existirem na literatura, as atuais técnicas baseadas em aprendizado de máquina são raramente utilizadas em produção [Smith et al. 2020], pois exigem a análise de múltiplos arquivos para classificação.

Assim, este artigo apresenta e descreve a ferramenta APKAnalyzer para classificação de arquivos APK. Trata-se de uma ferramenta que utiliza modelos de aprendizado de máquina previamente treinados com a abordagem *multi-view* e seleção de características com otimização multiobjetivo a partir de um novo *dataset*. Com isso, aprimoramos as taxas de verdadeiro positivo em uma média de 4,4, exigindo até 65% dos custos com processamento de inferência.

Em resumo, as principais contribuições desta pesquisa que se refletem no motor de classificação da ferramenta APKAnalyzer são:

- Um novo *dataset multi-view* de *malware* para *Android* com 3 conjuntos de características complementares (permissões, *opcodes* e chamadas de API), o qual foi construído com 40 mil arquivos de APK;
- Um novo modelo de detecção de *malware* para *Android multi-view* implementado com uma estratégia de otimização multiobjetivo.

Este artigo estrutura-se da seguinte forma: a seção 2 apresenta o artigo de pesquisa relacionado a esta ferramenta; a seção 3 descreve a arquitetura e detalhes técnicos da ferramenta; a seção 4 apresenta uma demonstração de utilização da ferramenta; a seção 5 indica o repositório do GitHub para acesso; e a seção 6 sumariza as conclusões deste trabalho e próximos passos no desenvolvimento da ferramenta.

2. Artigo Relacionado

Esta ferramenta foi idealizada e desenvolvida na esteira do artigo que foi submetido à apreciação no XVIII Workshop de Trabalhos de Iniciação Científica e de Graduação (WTICG) do 24º Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais. Trata-se de um artigo que contribui na discussão do problema de detecção de *malware* para *Android* em um cenário *multi-view* utilizando otimização multiobjetivo.

O modelo proposto considera um fluxo de processamento para classificação baseado em aprendizado de máquina com *multi-view*, que inicia após a análise estática de arquivos APK. O arquivo APK *Android* é analisado através de um processo *multi-view*, em que são utilizados vários arquivos para a tarefa de classificação. Por exemplo, analisando os arquivos *manifest.xml*, onde encontram-se as permissões, e o arquivo *dex*, responsavél pelos *opcodes* e as chamadas de API. Trata-se de uma abordagem que aprimora

a generalização e a confiança do processo de classificação. O comportamento de cada arquivo é extraído por um módulo de extração de características. Analogamente, os vetores de características resultantes são classificados por um classificador, produzindo um resultado de classificação para cada *view*. Para combinar as múltiplas classificações, utilizamos um módulo de combinação que efetua uma votação por maioria para produzir o resultado final da classificação.

A otimização multiobjetivo entra em cena como uma abordagem para enfrentar o desafio de combinar várias classificações com base em *views* complementares. Com isso, atingimos o objetivo de otimizar as características utilizadas por cada *view* com base na precisão de classificação combinado com os custos computacionais, viabilizando um processo de classificação com *multi-view*.

Ademais, foi criado um novo *dataset* com um total de 40 mil arquivos APK, os quais foram obtidos da plataforma AndroZoo [Allix et al. 2016]. E as características de cada view dos arquivos APK foram extraídas com a ferramenta AndroPyTool [Martín et al. 2019]. Adicionalmente, um módulo intermediário padroniza as características extraídas pelo AndroPyTool de forma a torná-las adequadas para as tarefas do módulo de classificação associado à aprendizagem de máquina.

3. Descrição da Ferramenta

A ferramenta proposta neste artigo, chamada de APKAnalyzer, possui um motor de classificação de arquivos APK que incorpora a abordagem descrita na seção 2.

A ferramenta APKAnalyzer foi desenvolvida na linguagem Python 3.10, utilizando como principais bibliotecas: `pandas v2.2` para extração das *views* a partir das *features* dos arquivos APK a serem analizados (permissões, *opcodes* e chamadas de API); `numpy v1.26` para a criação dos vetores comportamentais de cada *view*, os quais serão as entradas dos modelos de aprendizagem de máquina; `scikit-learn v1.4` para as tarefas de preprocessamento dos vetores comportamentais (ajuste de escala com o módulo `MinMaxScaler` e redução de dimensionalidade com o módulo `PCA`) e predição com os módulos `KNeighborsClassifier`, `RandomForestClassifier` e `DecisionTreeClassifier`; e `joblib v1.4` para carregar os módulos utilizados na fase de treinamento dos modelos. Além dessas bibliotecas, a ferramenta necessita do AndroPyTool [Martín et al. 2019] para extração das *features* dos arquivos APK no processo de análise estática. AndroPyTool, por sua vez, necessita da plataforma de contêineres Docker. Acerca das dependências, a versão inicial da ferramenta não automatiza a instalação delas, embora faça uma verificação inicial e apenas inicialize se todas as dependências estiverem resolvidas.

A ferramenta foi construída com uma interface de linha de comando, que recebe como argumento um arquivo APK ou um diretório. O processamento é feito sequencialmente seguindo as etapas: inicialização da aplicação, no qual é verificada as dependências da ferramenta e o argumento de entrada da linha de comando; obtenção do(s) arquivo(s) APK, no qual o(s) arquivo(s) é (são) validado(s) e copiado(s) para um diretório interno da aplicação; extração das *features* por meio de análise estática; extração e estruturação das *views*, no qual é gerada a estrutura de *features* por *view* de cada arquivo APK baseado em uma estrutura padrão seguindo a ordem utilizada na fase de treinamento; preprocessamento, no qual é aplicada a normalização, redução de dimensão e seleção das *features*

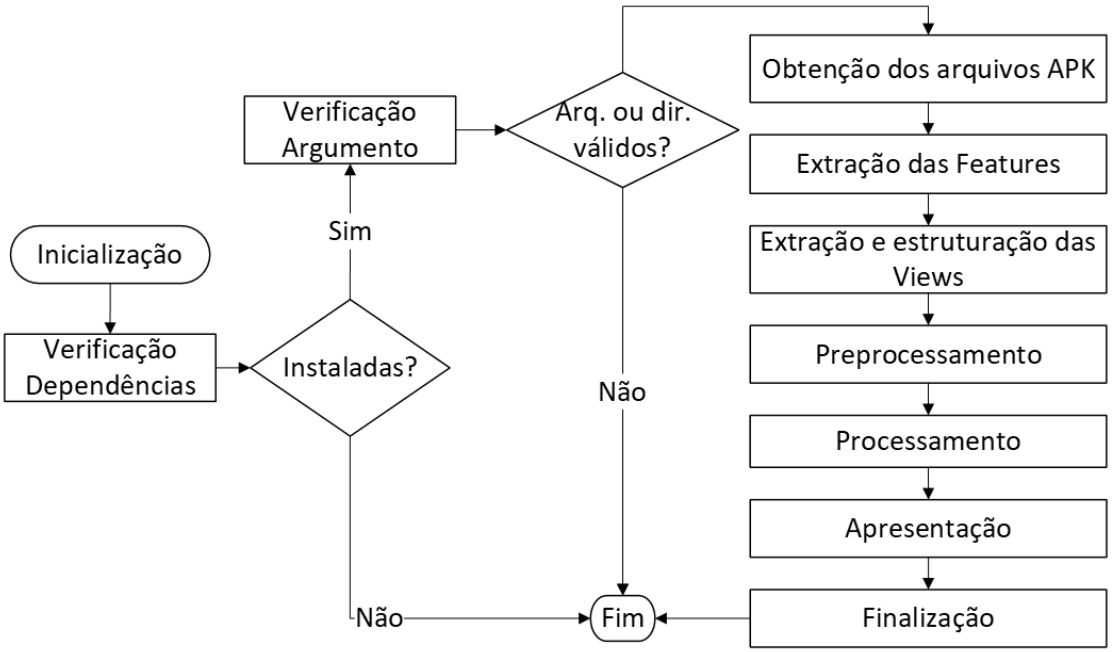


Figura 1. Fluxograma dos processos da ferramenta APKAnalyzer.

obtidas previamente na fase de treino dos modelos; processamento, no qual é feita a predição baseada em voto majoritário a partir dos resultados de classificação dos modelos para cada *view*; apresentação, no qual é apresentado os resultados da classificação; e finalização, no qual é feita a exclusão do(s) arquivo(s) copiado(s) na etapa de obtenção.

3.1. Inicialização da aplicação

No módulo de inicialização, a ferramenta verifica se todas as dependências estão adequadas e se o argumento recebido é válido. Caso alguma das dependências não seja encontrada ou o argumento é inválido, a execução da aplicação é abortada, informando qual dependência deve ser instalada, ser for o caso, ou que o argumento é inválido.

Sobre a verificação das dependência, a ferramenta analisa se a plataforma de contêineres Docker está instalada e se foi configurada para não solicitar elevação de privilégio (*non-root user*). Além disso, é verificado se a imagem `alexmyg/andropytool:latest` está disponível. Com exceção dessa última dependência, as duas anteriores precisam ser feitas pelo usuário da ferramenta. Em seguida, é verificado se os arquivos serializados previamente na fase de treinamento dos modelos estão disponíveis. Além desses arquivos, a ferramenta verifica se os arquivos com a estrutura de *features* de cada *view* está acessível. Existem dois diretórios no diretório raiz da aplicação que armazenam esses arquivos, conforme descrito a seguir:

- `./dumps`: armazena os arquivos serializados. Arquivos para normalização, por exemplo: `apicalls_minMaxScaler.pkl`; Arquivos para redução de dimensionalidade, por exemplo: `apicalls_pca.pkl`; Arquivos para seleção de `features`, por exemplo: `feature_selection_knn.pkl`; Arquivos para predição, por exemplo: `apicalls_dt_pca_las.pkl`;
- `./schemas`: armazena arquivos que representam a estrutura de todas as *features* de cada *view* antes da normalização e dimensionamento.

Por fim, na fase das dependências, é verificado se as bibliotecas da linguagem Python estão instaladas na versão correta: `joblib v1.4`, `numpy v1.26`, `pandas v2.2`, `scikit-learn v1.4`, `tabulate v0.9`.

Em relação ao argumento passado para aplicação, espera-se receber um arquivo de APK ou um diretório válido, conforme exemplo: `pyhton apkanalyzer arquivo_apk.apk` ou `pyhton apkanalyzer dir_apks`.

3.2. Obtenção do(s) arquivo(s) APK

Nesta etapa, a ferramenta faz uma cópia do(s) arquivo(s) indicado(s) no argumento, que a aplicação recebe pela linha de comando, para o diretório `./apks`, localizado na raiz do diretório da aplicação. Optou-se pela cópia para garantir as permissões necessárias para execução da ferramenta AndroPyTool. Os arquivos permanecem no diretório `./apks` até a conclusão da execução da aplicação, sendo excluídos caso o usuário que executa a ferramenta tenha os privilégios adequados.

3.3. Extração das *features*

O módulo de extração das *features* é responsável por inicializar a ferramenta AndroPyTool, utilizando o comando: `docker run --volume ./apks:/apks alexmyg/andropytool -s /apks/ -fw`. Com isso, é feita uma análise estática dos arquivos APK que estivem no diretório `./apks`. AndroPyTool produz como resultado alguns diretório, dentro os quais `./apks/Features_files/` conterá os arquivos com todas as *features* de cada um dos arquivos APK processados.

3.4. Extração e estruturação das *views*

Os arquivos de *features* resultantes da análise estática realizada pela ferramenta AndroPyTool contêm todas as *features* de um arquivo APK, além das que são utilizadas pelo nosso modelo de classificação, quais sejam, permissões, *opcodes* e chamadas de API. Por isso, nesta etapa, o módulo de extração e estruturação das *views* extrai de cada arquivo de*feature*, gerado pala ferramenta AndroPyTool, por arquivo de APK, as três views. Para manter a consistência dos modelos de classificação, é preciso estruturar cada uma das *views* extraídas, ordenando as *features* de acordo com o mesmo modelo que foi utilizado previamente no treinamento dos classificadores. Para isso, a ferramenta traz três *schemas* de estrutura para cada uma das *views* disponíveis no diretório *./schemas*. Assim, esse *schema* serve de modelo para estruturar e ordenar as *features* em cada view, se adequando ao que é esperado nas próximas etapas.

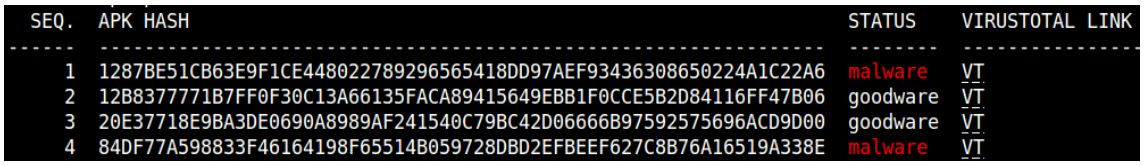
3.5. Preprocessamento

Ao chegar nesta etapa, a ferramenta possui uma lista com as APK, identificadas pelo hash, a serem classificadas e as suas respectivas características de cada uma das três *views* já estruturadas de acordo com os *schemas*. Com isso, nesta etapa, os valores das características são normalizados, aplicando um algoritmo de *minmaxscaler*, e a dimensão do vetor de características é reduzida para 100 características, utilizando um algoritmo PCA.

Por fim, aplica-se a seleção de características em cada *view* com base em um conjunto ideal de características gerado pelo algoritmo genético multiobjetivo NSGA-II

Tabela 1. Exemplo do voto majoritário no processo de predição.

	Permissões	*Opcodes*	Chamadas de API	Resultado Parcial
DT	0	0	1	*Goodware*
RF	1	1	0	*Malware*
kNN	1	1	1	*Malware*
Resultado Final				Malware



SEQ.	APK HASH	STATUS	VIRUSTOTAL LINK
1	1287BE51CB63E9F1CE4480227892965654 18DD97AEF93436308650224A1C22A6	malware	VT
2	12B8377771B7FF0F30C13A66135FACA89415649EBB1F0CCE5B2D84116FF47B06	goodware	VT
3	20E37718E9BA3DE0690A8989AF241540C79BC42D06666B97592575696ACD9D00	goodware	VT
4	84DF77A598833F46164198F65514B059728DBD2EFBEEF627C8B76A16519A338E	malware	VT

Figura 2. Exemplo da tela de visualização dos resultados.

(Non-dominated Sorting Genetic Algorithm II), criado na fase de treinamento dos modelos. Esse conjunto ideal foi serializado e está disponível no diretório `./dumps`, consistindo de três vetores, um para cada *view*, com cada elemento do vetor contendo um número entre 0 e 1. Se o número for menor que 0.5, a característica deve ser excluída.

3.6. Processamento

O modelo de processamento recebe a lista de todos os arquivos APK, representados pelo identificador hash, e contendo a estrutura de *views* com suas respectivas features já normalizadas e dimensionadas. É iniciado, então, a etapa de predição, utilizando três modelos de classificação, quais sejam, RF, DT, kNN. Para cada um dos modelos, é feita a predição utilizando cada uma das três views, totalizando 9 predições por identificador hash do arquivo de APK. O voto majoritário é utilizado para obter resultados parciais e o resultado final, considerando *goodware* se a soma do resultado dos classificadores for menor que 2, caso contrário, *malware*, conforme exemplo na tabela 1

3.7. Apresentação

A etapa de apresentação consiste em exibir os resultados obtidos no terminal, apresentando: um número sequencial para indexar o resultado, o hash identificador do arquivo APK, a classificação, e um link para o Vírus Total; conforme exemplo na figura 2. A última etapa do fluxo de processo da aplicação APKAnalyzer realiza a exclusão dos arquivos copiados para processamento, caso o usuário da aplicação detenha privilégios para isso.

4. Demonstração da Ferramenta

Após o download da aplicação pelo repositório 5, é necessário seguir as orientações disponíveis no arquivo `README.md` para a instalação dos pré-requisitos da aplicação. Além disso, esse arquivo possui mais informações e demonstração sobre a aplicação. Atendidos os pré-requisitos, basta abrir o terminal no diretório raiz da aplicação e executar o comando: `python APKAnalyzer.py ./para-processar/6BE157CDE54C[...].apk`. Nesse exemplo, utiliza-se um arquivo de teste disponíveis no diretório `./para-processar/`, caso fosse desejado, poderia se passar um diretório para a aplicação, como no exemplo `python`

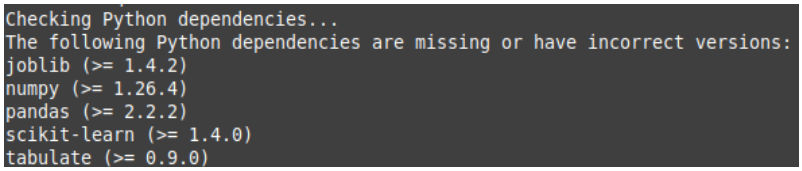


Figura 3. Exemplo da tela com falha na verificação de dependências.

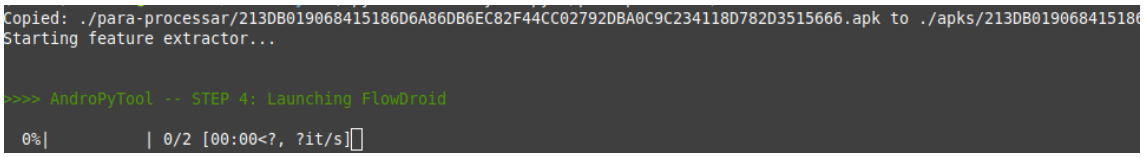


Figura 4. Exemplo do início do processamento.

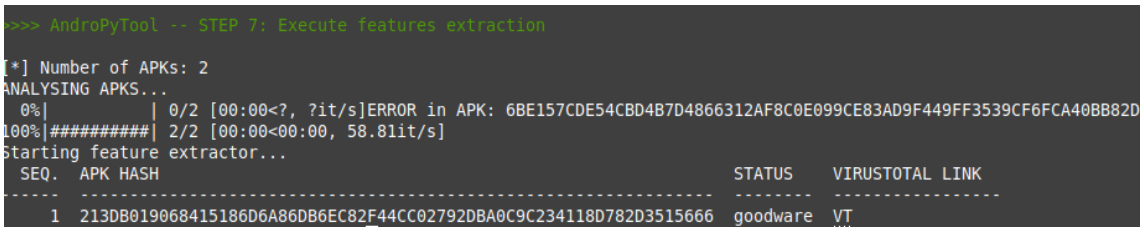


Figura 5. Exemplo do início do processamento.

`APKAnalyzer.py ./para-processar/`. Nesse caso, todos os arquivos no diretório serão processados, enquanto naquele, apenas o arquivo indicado, será copiado.

Na primeira execução da aplicação, um processo de verificação de dependências será iniciado, prosseguindo com o processamento apenas quando todas as dependências estiverem satisfeitas. Na Figura 3, temos um exemplo de quando a aplicação não encontra as bibliotecas python. Quando as dependências forem satisfeitas, a aplicação inicia o fluxo de processamento da aplicação, conforme descrito na seção 3. Na figura 4, destaca-se o processo de cópia dos dados de origem para o diretório `./apks` e, em seguindo, o início do processamento do AndroPyTool. Na Figura 5, destaca-se o resultado final do processamento do AndroPyTool, fase de extração das *features*. Caso a ferramenta AndroPyTool não consiga extrair as *features* de alguma APK, uma mensagem de erro é mostrada com o identificador do arquivo APK e ela não prosseguirá no fluxo de processamento. Além disso, destaca-se o resultado final do processamento da ferramenta APKAnalyzer.

5. Disponibilização da Ferramenta

A ferramenta APKAnalyzer está disponível em https://github.com/pFransozi/APKAnalyzer. Nesse repositório estão disponíveis: código fonte, manuais de uso e demonstração.

6. Conclusão

Neste trabalho foi apresentada a ferramenta APKAnalyzer. O objetivo dessa ferramenta foi implementar um modelo de detecção de malware para Android baseado em aprendizagem de máquina, incorporando duas técnicas: vetor comportamental da aplicação baseado em *multi-view* e seleção de *features* baseado em algoritmo genético com otimização multiobjetivo. A evolução da aplicação está interligada com melhorias no motor de

classificação, que resultem em uma acurácia maior em um menor tempo. No futuro, espera-se realizar teste em modelos de aprendizagem profunda.

Agradecimentos

Os autores agradecem ao CNPq pelo apoio financeiro parcial ao projeto, processos 304990/2021-3, 302937/2023-4, e 407879/2023-4.

Referências

Allix, K., Bissyandé, T. F., Klein, J., and Traon, Y. L. (2016). Androzoo: Collecting millions of android apps for the research community. *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*, pages 468–471.

AndroidStats. Android statistics (2024). `https://www.businessofapps.com/data/android-statistics/`. [online: acessado em 02-junho-2024].

Darwaish, A. and Nait-Abdesselam, F. (2020). Rgb-based android malware detection and classification using convolutional neural network. In *IEEE Global Communications Conference*.

dos Santos, R. R., Viegas, E. K., and Santin, A. O. (2021). A reminiscent intrusion detection model based on deep autoencoders and transfer learning. In *2021 IEEE Global Communications Conference (GLOBECOM)*. IEEE.

dos Santos, R. R., Viegas, E. K., Santin, A. O., and Tedeschi, P. (2023). Federated learning for reliable model updates in network-based intrusion detection. *Computers amp; Security*, 133:103413.

Geremias, J., Viegas, E. K., Santin, A. O., Britto, A., and Horchulhack, P. (2022). Towards multi-view android malware detection through image-based deep learning. In *2022 International Wireless Communications and Mobile Computing (IWCMC)*. IEEE.

Geremias, J., Viegas, E. K., Santin, A. O., Britto, A., and Horchulhack, P. (2023). Towards a reliable hierarchical android malware detection through image-based cnn. In *2023 IEEE 20th Consumer Communications amp; Networking Conference (CCNC)*. IEEE.

Horchulhack, P., Viegas, E. K., Santin, A. O., Ramos, F. V., and Tedeschi, P. (2024). Detection of quality of service degradation on multi-tenant containerized services. *Journal of Network and Computer Applications*, 224:103839.

Kaspersky. Attacks on mobile devices significantly increase in 2023. `https://www.kaspersky.com/about/press-releases/2024_attacks-on-mobile-devices-significantly-increase-in-2023`. [online: acessado em 02-junho-2024].

Martín, A., Lara-Cabrera, R., and Camacho, D. (2019). Android malware detection through hybrid features fusion and ensemble classifiers: The andropytool framework and the omnidroid dataset. *Information Fusion*, 52:128–142.

Smith, M. R., Johnson, N. T., Ingram, J. B., Carbajal, A. J., Haus, B. I., Domschot, E., Ramyaa, R., Lamb, C. C., Verzi, S. J., and Kegelmeyer, W. P. (2020). Mind the gap: On bridging the semantic gap between machine learning and malware analysis. In *Proceedings of the 13th ACM Workshop on Artificial Intelligence and Security*, CCS '20. ACM.