Joner de Mello Assolin

O impacto da seleção de permissões na detecção de malwares Android

Alegrete

2021

Joner de Mello Assolin

O impacto da seleção de permissões na detecção de malwares Android

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de software da Universidade Federal do Pampa como requisito parcial para a obtenção do título de Bacharel em Engenharia de software.

Universidade Federal do Pampa

Orientador: Prof. Dr. Diego Kreutz

Coorientador: Bel. Gustavo Cardozo Rodrigues

Alegrete

2021

Joner de Mello Assolin

O impacto da seleção de permissões na detecção de malwares Android

Trabalho de Conclusão apresentado ao Curso de Engenharia de Software da Universidade Federal do Pampa, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Trabalho de Conclusão de Curso defendido e aprovado em: 01 de outubro de 2021.

Banca examinadora:

_____

Prof. Diego Kreutz

Orientador

UniPampa

_____

Bel. Gustavo Cardozo Rodrigues

Coorientador

Combate à Fraude

_____

Prof. Dr. Eduardo Luzeiro Feitosa

UFAM

Esp. Vanderson da Silva Rocha

UFAM

Prof. Dr. Rodrigo Brandão Mansilha

UniPampa

Assinado eletronicamente por **Eduardo Luzeiro Feitosa**, **Usuário Externo**, em 01/10/2021, às 15:29, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

Assinado eletronicamente por **Vanderson da Silva Rocha**, **Usuário Externo**, em 01/10/2021, às 15:29, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

Assinado eletronicamente por **Gustavo Cardozo Rodrigues**, **Usuário Externo**, em 01/10/2021, às 15:29, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

Assinado eletronicamente por **DIEGO LUIS KREUTZ**, **PROFESSOR DO MAGISTERIO SUPERIOR**, em 01/10/2021, às 15:30, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

Assinado eletronicamente por **RODRIGO BRANDAO MANSILHA**, **PROFESSOR DO MAGISTERIO SUPERIOR**, em 01/10/2021, às 15:30, 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 **0625169** e o código CRC **AE4E1450**.

Este trabalho é dedicado a todas as pessoas
que sempre acreditaram no meu potencial
e contribuíram com essa conquista.

AGRADECIMENTOS

Agradeço a minha mãe Janete, ao amigo João Leonel Ferreira, pelos “puxões de orelha” e incentivo ao estudo. A Tábata que é a maior incentivadora e pela paciência e compreensão nos momento de estresse.

Aos professores, pelas correções e ensinamentos que me permitiram apresentar um melhor desempenho no meu processo de formação profissional. Em especial aos amigos e orientadores, Diego Kreutz e Gustavo Cardozo, que acreditaram em mim, e me ajudaram a extrair o meu potencial. Muito obrigado pela paciência.

Aos colegas do projeto Malware-Hunter que foram fundamentais na tomada de decisão para a escolha do tema do trabalho e ajuda no que foi preciso. Também a todos que direta ou indiretamente fizeram parte da minha formação, o meu muito obrigado.

“A ciência trabalha na fronteira entre conhecimento e a ignorância.
Não temos medo de admitir o que não sabemos,
não há vergonha nisso. A única vergonha é
fingir que temos todas as respostas.“
(Neil deGrasse Tyson).

RESUMO

O número de aplicativos maliciosos vem crescendo rapidamente na plataforma Android. Atualmente, muitas pesquisas utilizam modelos preditivos de aprendizado de máquina para detecção de *malwares* Android. Para atacar o desafio de escalabilidade derivado deste contexto, há trabalhos que propõe a utilização de um número reduzido de permissões, como é o caso do SigPID. Neste trabalho, apresentamos um levantamento das permissões Android mais frequentemente utilizadas em métodos de aprendizagem de máquina e a realização da reprodução do SigPID, incluindo os 3 níveis de seleção de permissões e implementação e avaliação dos principais métodos de aprendizagem. Diferentemente do trabalho original, a reprodução foi realizada utilizando um conjunto de dados publicamente disponível. Nós comparamos as permissões em cada nível de seleção com as 32 permissões identificadas como mais recorrentes; as 113 do conjunto de dados público escolhido e as 22 permissões (contidas no conjunto de dados), consideradas perigosas pela Google. Nosso estudo inicial indica que o número de permissões impacta o tempo de treinamento e execução, bem como a acurácia dos modelos. Entretanto, o tempo de execução pode não ser significativo a ponto de justificar um número menor de permissões para detecção de *malwares* (*e.g.*, redução de 113 permissões para 27).

Palavras-chave: Aprendizado de Máquina. Malwares. Permissões. Android.

ABSTRACT

The number of malicious apps is growing rapidly on the Android platform. Currently, many researches use predictive machine learning models to detect Android *malware*. To tackle the scalability challenge derived from this context, there are works that propose the use of a reduced number of permissions, as is the case with SigPID. In this work, we present a survey of the Android permissions most frequently used in machine learning methods and the performance of SigPID reproduction, including the 3 levels of permission selection and implementation and evaluation of the main learning methods. Unlike the original work, the reproduction was performed using a publicly available dataset. We compared the permissions at each selection level with the 32 permissions identified as the most recurring; the 113 from the chosen public dataset and the 22 permissions (contained in the dataset) considered dangerous by Google. Our initial study indicates that the number of permissions impacts training and execution time, as well as the accuracy of models. However, the execution time may not be significant enough to justify a smaller number of permissions for malware detection (*e.g.*, reduction from 113 permissions to 27).

Key-words: Machine Learning. Malwares. Permissions. Android.

SUMÁRIO

1 INTRODUÇÃO

Dentre os métodos para detecção de *malwares* em aplicativos Android, os que utilizam aprendizado de máquina vêm ganhando destaque (BAYAZIT; SAHINGOZ; DOGAN, 2020; WU; ZHU; LIU, 2021). Independente de focarem suas abordagens na análise estática, dinâmica ou híbrida (SHARMA; RATTAN, 2021; GYAMFI; OWUSU, 2018), esses trabalhos utilizam as permissões do Android para o desenvolvimento de modelos de detecção de *malwares* com bom desempenho (ALSOGHYER; ALMOMANI, 2020). Entretanto, utilizar todas as 247 permissões das APIs do Android (Google Developer, 2021), disponíveis para treinamento dos modelos de aprendizado de máquina, pode representar um desafio de escalabilidade (CHAKKARAVARTHY; SANGEETHA; VAIDEHI, 2019; Li et al., 2018) e impactar negativamente no tempo de execução das soluções.

Com o objetivo de mitigar o problema da escalabilidade, há trabalhos (*e.g.*, (Li et al., 2018; YILDIZ; DOĞRU, 2019)) que investigaram o impacto da redução dos números de permissões utilizadas para o treino dos modelos. Como resultado, verificaram que, mesmo utilizando um número menor de permissões (*i.e.* 22), o tempo de execução pode ser reduzido sem comprometer de forma significativa a acurácia do modelo (*i.e.*, aumento da escalabilidade sem comprometer o desempenho da classificação).

Neste trabalho, apresentamos três atividades desenvolvidas com foco em reduzir o número de permissões do Android (*i.e.*, número de *features*) para detecção de *malwares*. Na primeira atividade (ASSOLIN et al., 2021), realizamos um levantamento bibliográfico de trabalhos que discutiam permissões recorrentes em *malwares* do Android e que aplicavam aprendizado de máquina para detecção de aplicativos maliciosos. Ao todo, foram selecionados nove trabalhos que categorizavam essas recorrências. Com base nesses estudos, realizamos a intersecção entre as permissões recorrentes e chegamos ao total de 32 permissões, como detalhado em (ASSOLIN et al., 2021).

Na segunda atividade avaliamos e discutimos a reprodutibilidade e o desempenho do SigPID (Li et al., 2018; SUN et al., 2016), que pode ser considerado um dos mais relevantes e mais bem citados (mais de 320 citações GSC em setembro de 2021) sobre escalabilidade de modelos de detecção de *malwares* Android. Nesta atividade, reproduzimos a abordagem *Multi Level Data Pruning* (MLDP) (Poda Multinível de Dados) utilizada no SigPID, que consiste em três níveis de seleção de permissões. Esses níveis são utilizados para reduzir o número total de permissões empregadas no treinamento dos modelos, selecionando as mais relevantes.

Após aplicar o MLDP, avaliamos o desempenho do modelo em comparação com diferentes conjuntos de permissões, a saber: (a) 27 permissões selecionadas com a aplicação

do modelo MLDP; (b) 113 permissões (*baseline*) contidas no conjunto de dados público escolhido; (c) 22 permissões identificadas no trabalho original do SigPID; (d) 32 permissões mais recorrentes em trabalhos de detecção de *malwares* Android; e (e) 22 permissões classificadas perigosas pela Google[1]. Finalmente, na última atividade, implementamos também uma API Web [2] que disponibiliza recursos para análise estática de aplicativos Android (APKs), indo desde a extração das permissões até a execução dos modelos para a classificação do APK.

Como contribuição do trabalho, podemos destacar:

- Mapeamento e identificação das 32 permissões mais recorrentes em trabalhos de detecção de *malwares* que baseiam-se em permissões (ASSOLIN et al., 2021);
- Análise de aspectos de reprodutibilidade do SigPID;
- Reprodução da estratégia de seleção de permissões empregada pelo SigPID em um conjunto de dados público;
- Identificação de um subconjunto reduzido de permissões significativas que pode ser utilizado para identificar *malwares* no Android;
- Análise e discussão sobre o desempenho *versus* o tempo de execução de diferentes modelos de aprendizado de máquina;
- Implementação de uma ferramenta Web para análise estática de *malwares* baseada em permissões significativas.

O trabalho está organizado como segue. Nos Capítulos 2 e 3 apresentamos uma visão geral sobre os requisitos para a reprodução do SigPID e detalhamos a metodologia de seleção das permissões, respectivamente. Nos Capítulos 4 e 5 discutimos os resultados e a implementação da ferramenta. Por fim, nos Capítulos 6 e 7 pontuamos algumas discussões e apresentamos as considerações finais.

[1] Ao total, a Google define 30 permissões como perigosas, porém apenas 22 estão presentes no conjunto de dados utilizado.

[2] <https://github.com/Malware-Hunter/SigPID>

2 AMBIENTE E CONJUNTO DE DADOS

2.1 Detalhamento do Ambiente

Para o desenvolvimento e avaliação dos experimentos, utilizamos um notebook com processador Intel Celeron 1007U (1.5GHz, Dual Core, 2MB L2), 4GB DDR3 1.600MHz, disco rígido de 320GB (SATA - 5.400rpm), Windows 10 Home Single Language, compilação 19042.1110. Para a implementação e avaliação dos modelos, utilizamos as ferramentas Jupyter Notebook (IPython 7.12.0, Python 3.7.6 (default, jan. 8 2020) e o Google Chrome Versão 91.0.4472.124 (Versão oficial) 64 bits. Com exceção do algoritmo *Funtional Tree*, versão 1.0.4, implementado com a ferramenta Weka versão 3.9.5, os demais foram implementados utilizando a versão 0.22.1 da biblioteca Scikit-learn.

Para análise e uso do *dataset*, utilizamos uma divisão estratificada pseudo-aleatória (*test_size*) de 70%/30% (JAMES et al., 2013), a partir dos dados iniciais, sendo 70% utilizado para treinos e 30% para testes. As divisões são desejáveis em casos de conjunto de dados desbalanceados, como é o caso do conjunto de dados escolhido.

Para garantir a reprodutibilidade do experimento, definimos arbitrariamente a semente aleatória como 1 para `train_test_split`, de forma a controlar a seleção dos dados de treino e teste. Já os hiperparâmetros, variáveis que controlam o próprio processo de treinamento, foram seguidos conforme o padrão da biblioteca Scikit-learn. Esses parâmetros estão detalhados na pagina github[1], juntamente com os códigos fontes utilizados nos experimentos.

2.2 Conjunto de dados

Como mencionado, o conjunto de dados original do trabalho está indisponível. Para a reprodução e comparação do SigPID com diferentes conjuntos de permissões, selecionamos o conjunto de dados Drebin_215 (YERIMA; SEZER, 2018), um sub conjunto do *Drebin project* (ARP et al., 2014), por ser de acesso público e possuir permissões do Android como atributos. O Drebin_215, disponível publicamente no FigShare [2], possui 215 atributos extraídos de 15.036 aplicativos (5.560 malignos e 9.476 benignos), sendo que 113 atributos são permissões.

[1] <https://github.com/Malware-Hunter/SigPID>

[2] <https://figshare.com/articles/dataset/Android_malware_dataset_for_machine_learning_2/5854653>

3 MLDP: REMOÇÃO MULTINÍVEL DE DADOS

MLDP (*Multi-Level Data Pruning*) é um método multinível (de 3 níveis) para seleção de características cujo objetivo é reduzir o número de permissões de um conjunto de dados para o treino dos modelos de aprendizado de máquina. A ideia por trás do método é diminuir significativamente o número de permissões e, consequentemente, o tempo de execução dos modelos (Li et al., 2018). O MLDP assume como parâmetro de seleção uma taxa de acurácia e precisão de no mínimo 90%, considerada uma taxa ótima.

O MLDP opera nos seguintes três níveis de seleção: (1) classificação de permissão com taxa negativa (*Permission Ranking Negative Rate* ou PRNR); (2) classificação de permissão baseada em suporte (*Support Based Permission Ranking* ou SPR); e (3) mineração de permissões com regras de associação (*Permission Mining with Association Rules* ou PMAR). Cada um dos três níveis é detalhado a seguir.

3.1 PRNR: classificação de permissão com taxa negativa

A PRNR opera em duas matrizes, uma de permissões utilizadas por amostras de *malwares* e outra utilizada por aplicativos benignos, onde cada linha corresponde a um aplicativo e cada coluna a uma permissão. O objetivo é remover as permissões que são frequentemente solicitadas tanto por aplicativos maliciosos quanto por benignos (*e.g.*, INTERNET, CAMERA, BLUETOOTH).

Como o número de aplicativos benignos tende a ser maior que o de *malwares* em um conjunto de dados, o PRNR do SigPID propõe a equação 3.1 para equilibrar às duas matrizes. A equação calcula o suporte de cada permissão no conjunto de dados maior e, em seguida, dimensiona proporcionalmente o suporte para corresponder ao conjunto de dados menor. O suporte é a frequência com que cada permissão aparece no conjunto de dados.

$$S_B(P_j) = \frac{\sum_i B_{ij}}{Size(B_j)} * Size(M_j) \tag{3.1}$$

Na equação, M representa a matriz de permissões dos aplicativos maliciosos e B dos benignos. (P_j) denota permissão e $S_B(P_j)$ representa o suporte da permissão na matriz B. A listagem 3.1 apresenta o código da implementação da equação 3.1 utilizando a linguagem de programação Python.

Listagem 3.1 – Implementação da equação 3.1

```
def S_B(j):
    sigmaBij = B.sum(axis = 0, skipna = True)[j]
    sizeBj = B.shape[0]
    sizeMj = M.shape[0]
    return (sigmaBij/sizeBj)*sizeMj
```

A implementação da PRNR é baseada na equação 3.2, que determina uma classificação para cada permissão, variando no intervalo [-1, 1]. Na equação, se $R(P_j) = 1$, significa que a permissão (P_j) é apenas usada no conjunto de dados malicioso e que é uma permissão de alto risco. Se $R(P_j) = -1$, significa que a permissão (P_j) só é usada no conjunto de dados benigno e é uma permissão de baixo risco. Por outro lado, se $R(P_j) = 0$, significa que (P_j) é irrelevante na detecção de *malware* pois aparece em ambos os conjuntos de dado (*malwares* e benignos), gerando assim ambiguidade no modelo.

$$R(P_j) = \frac{\sum_i M_{ij} - S_B(P_j)}{\sum_i M_{ij} + S_B(P_j)} \tag{3.2}$$

O processamento da equação do primeiro nível de seleção, PRNR, é ilustrado na Figura 1. Ele foi implementado utilizando o código da listagem 3.2.

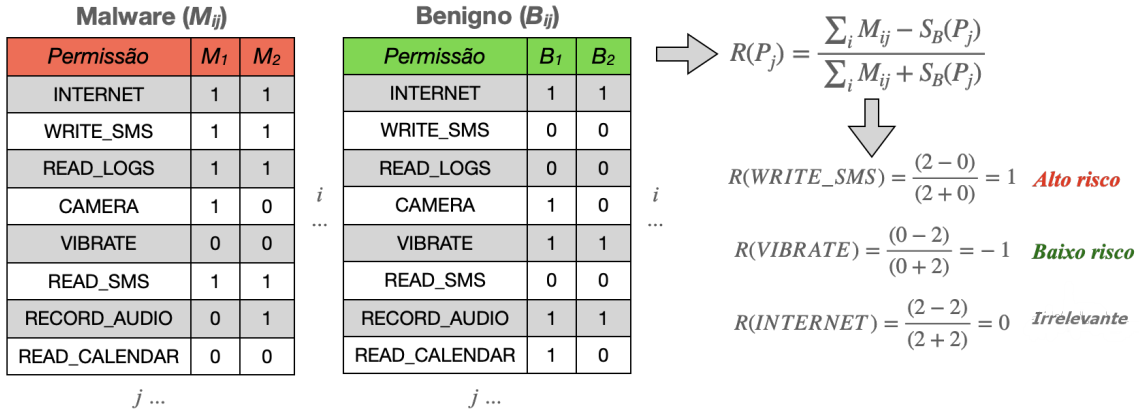

Figura 1 – Exemplo do cálculo do ranking de cada permissão

Listagem 3.2 – Implementação da equação 3.2

```
def PRNR(j):
    sigmaMij = M.sum(axis = 0, skipna = True)[j]
    S_Bj = S_B(j)
    return (sigmaMij-S_Bj)/(sigmaMij+S_Bj)
```

O próximo passo é ordenar a lista dos valores obtidos pelo PRNR, associando-a pela ordem crescente aos aplicativos benignos e pela ordem decrescente aos *malwares*, conforme

ilustrado na Figura 2. Dando continuidade ao processo de classificação, é utilizado o *Permission Incremental System* (PIS). As permissões são agrupadas 3 a 3, iniciando pelo topo de cada lista. A cada incremento de 6 permissões (3 benignos e 3 *malwares*), os grupos de permissões são submetidos ao algorítimo de aprendizado de máquina *Support Vector Machine* (SVM).

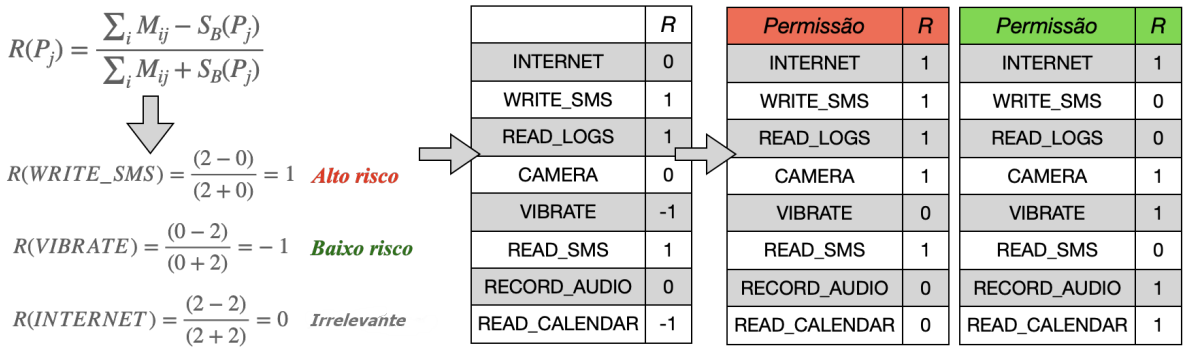

Figura 2 – Ordenação pelo ranking

A cada novo grupo, é avaliado o poder preditivo de detecção de *malware* utilizando as métricas (BOUTABA et al., 2018) descritas na Tabela 1.

Tabela 1 – Métricas

Acurácia $= \frac{TP+TN}{100}$	Precisão $= \frac{TP}{TP+FP}$
Recall $= \frac{TP}{TP+FN}$	F1_Score $= \frac{2*(Percisão*Recall)}{(Percisão+Recall)}$

O momento de seleção ocorre quando as métricas chegam ao seu valor máximo e, posteriormente, começam a decair. O objetivo do processo é encontrar o menor número de permissões que produza os melhores *scores* de detecção de *malware*. Como resultado do primeiro nível de seleção, chegamos a 108 permissões, de um total de 113 contidas no conjunto de dados. A Figura 3 ilustra o desempenho do PRNR para cada incremento (PIS).

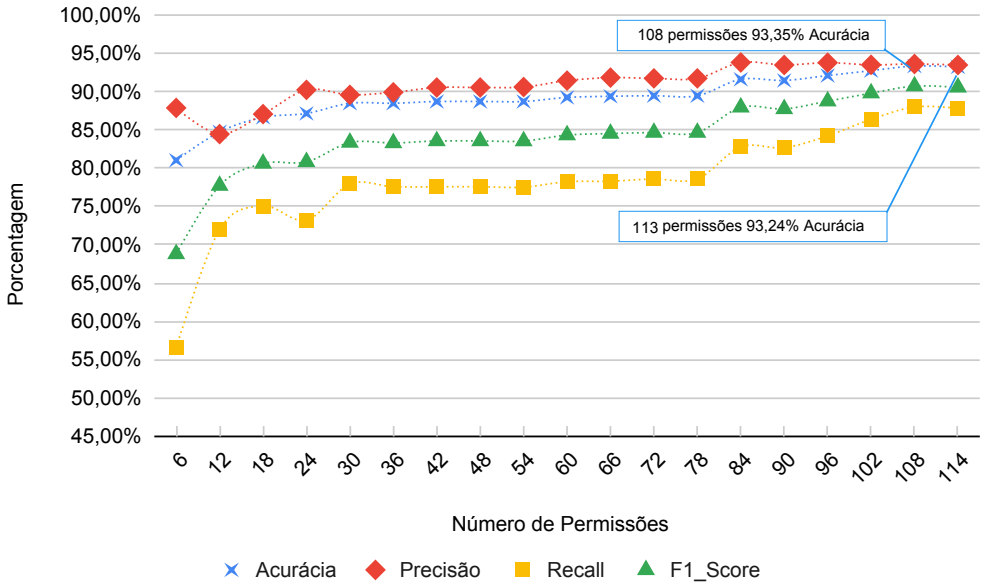

Figura 3 – Desempenho do PRNR a cada incremento (PIS)

Podemos observar no gráfico da Figura 3 que utilizando 108 permissões alcançamos acurácia de 93,35%. já com 113 permissões ocorre uma pequena redução na acurácia para 93,24%.

3.2 SPR: classificação de permissão baseada em suporte

A classificação de permissão baseada em suporte busca avaliar a recorrência de uma permissão. Se ela possuir uma baixa frequência, seu impacto será mínimo no desempenho da detecção de *malware*. As permissões consideradas de baixo desempenho são excluídas. As 108 permissões selecionadas no passo anterior (PRNR) são ordenadas em ordem decrescente de acordo com o seu suporte.

O objetivo da SPR é encontrar o menor número de permissões de alto suporte capaz de produzir uma acurácia de detecção acima de 90%. Novamente é aplicado o Sistema Incremental de Permissão (PIS), porém agora, de 5 em 5 permissões. Quando o modelo atingir 90% de acurácia, selecionamos as permissões contidas no incremento. Com 30 permissões foi possível satisfazer a condição e atingir 90,07% de acurácia, conforme podemos observar no gráfico da Figura 4.

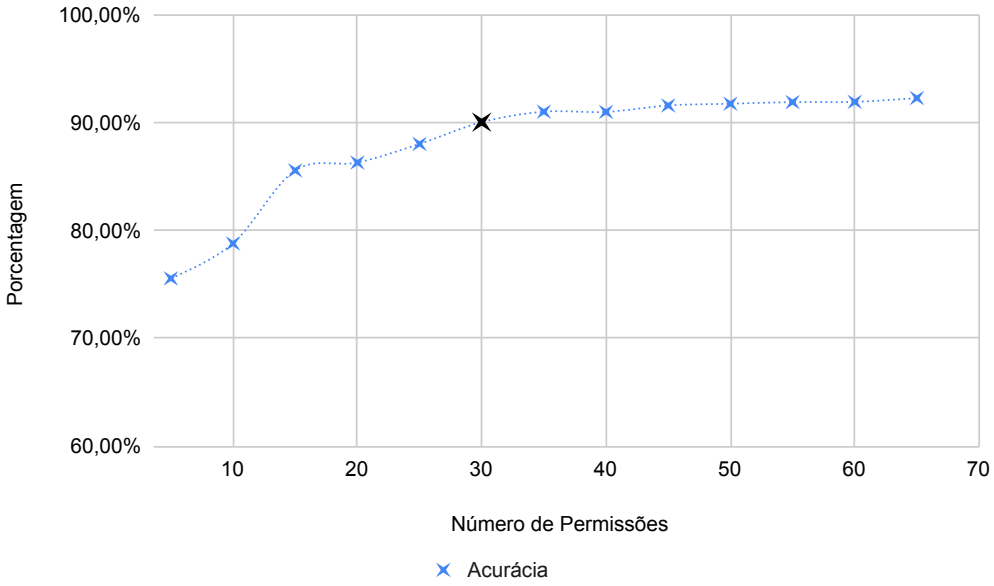

Figura 4 – Desempenho da SPR para cada incremento (PIS)

3.3 PMAR: mineração de permissões com regras de associação

O último nível de seleção é chamado mineração de permissões com regras de associação (PMAR) inspeciona permissões que possuem maior probabilidade de estarem associadas (*e.g.*, WRITE_SMS e READ_SMS). Então, a permissão com o menor suporte é eliminada.

Para identificar a relação de cada permissão, é aplicado o algorítimo de regas de associação Apriori (AGRAWAL; SRIKANT et al., 1994) com os parâmetros de 96,5% de confiança mínima e 10% de suporte mínimo, pois é o mesmo definido no trabalho original do SigPID. O Apriori calcula a probabilidade de um item estar presente em um conjunto de itens frequente, desde que outro item esteja presente.

A Tabela 2 representa a execução do Apriori sobre as 30 permissões selecionadas após as etapas PRNR e SPR. A confiança de uma regra indica a probabilidade do antecedente e o consequente aparecerem na mesma transação. Ou seja, é a probabilidade condicional do consequente dado o antecedente, por exemplo, CHANGE_WIFI_STATE implica ACCESS_WIFI_STATE com 99,3% de confiança. Já o suporte de uma regra indica a frequência com que os itens na regra ocorrem juntos. Por exemplo, CHANGE WIFI_STATE e ACCESS_WIFI_STATE podem aparecer juntos em 16,07% das transações. Nesse caso, as duas regras a seguir teriam, cada uma, um suporte mínimo de 16,07%. O *lift* nos diz que a probabilidade de WRITE_SMS e READ_SMS aparecerem juntos é 5,26 vezes maior do que a probabilidade de apenas READ_SMS e que são positivamente correlacionados.

Tabela 2 – Saída do algoritmo Apriori

Antecedentes	Consequentes	Suporte	Confiança	Lift
CHANGE_WIFI_STATE	ACCESS_WIFI_STATE	0.160758	0.993016	2.28
MANAGE_ACCOUNTS	GET_ACCOUNTS	0.103359	0.992971	3.32
WRITE_SMS	READ_SMS	0.111407	0.984136	5.26

Apesar de WRITE_SMS e READ_SMS pertencerem à lista de permissões perigosas da Google, não é necessário considerar ambas as permissões, pois uma delas é suficiente para caracterizar comportamentos de aplicativos maliciosos. Algo similar ocorre com outras permissões. Por exemplo, pode-se remover, além de WRITE_SMS, as permissões MANAGE_ACCOUNTS e ACCESS_WIFI_STATE. Após remover as 3 permissões, identificadas pela regra de associação, chega-se a um conjunto de 27 permissões, apresentadas na Tabela 3.

Tabela 3 – Lista de permissões selecionadas após aplicação dos três níveis de seleção do MLDP

ACCESS_NETWORK_STATE	CHANGE_WIFI_STATE	WRITE_EXTERNAL_STORAGE
WRITE_SETTINGS	READ_PHONE_STATE	CAMERA
WAKE_LOCK	CALL_PHONE	RECEIVE_BOOT_COMPLETED
WRITE_CONTACTS	VIBRATE	READ_EXTERNAL_STORAGE
GET_ACCOUNTS	USE_CREDENTIALS	ACCESS_FINE_LOCATION
READ_HISTORY_BOOKMARKS	ACCESS_COARSE_LOCATION	CHANGE_NETWORK_STATE
SEND_SMS	RECORD_AUDIO	READ_CONTACTS
READ_SYNC_SETTINGS	RECEIVE_SMS	RESTART_PACKAGES
READ_SMS	BLUETOOTH	GET_TASKS

4 RESULTADOS

Para a reprodução do SigPID, utilizamos o algoritmo SVM, conforme proposto pelos autores no trabalho original (Li et al., 2018). A Tabela 4 apresenta os resultados da execução do SVM para os diferentes conjuntos de dados, incluindo métricas para a avaliação do desempenho da detecção e o tempo de execução.

Tabela 4 – Métricas de avaliação SVM

Conjunto de Dados	Quantidade de Permissões	Precisão	Recall	FPR	F1_Score	Acurácia	Tempo Execução (s)
PRNR	108	93,62	88,01	3,52	90,73	93,35	5,44
PRNR+SPR	30	90,03	82,25	5,35	85,96	90,07	2,41
PRNR+SPR+PMAR	27	90,13	82,07	5,28	85,91	90,05	2,26
Baseline	113	93,49	87,83	3,59	90,57	93,24	5,84
Perigosas Google	22	86,76	71,88	6,44	78,62	85,55	2,34
Recorrentes	32	88,54	81,53	6,19	84,89	89,27	3,17
SigPID	22	91,77	74,88	3,94	82,47	88,23	2,62

Como podemos observar, os conjuntos de dados que utilizam o mesmo número de permissões obtiveram resultados diferentes. As 22 permissões identificadas no SigPID se destacam em todas as métricas de avaliação quando comparadas com as 22 permissões consideradas perigosas pela Google, ou seja, utilizar permissões perigosas não leva a um melhor resultado qualitativo. Isto indica que a escolha das permissões possui, de fato, um impacto no desempenho na detecção de *malwares*.

Testamos também o conjunto de dados que utiliza 32 permissões que mais aparecem em *malwares*. Elas foram identificadas por meio da intersecção entre permissões encontradas em 9 trabalhos na literatura, conforme detalhamos em (ASSOLIN et al., 2021). Podemos observar na Tabela 4 que, embora as 32 permissões recorrentes tenham atingido uma acurácia melhor em comparação a conjuntos de dados que utilizam 22 permissões (SigPID e Perigosas da Google), ainda assim, não atingem 90% de acurácia como o conjunto de dados PRNR+SPR+PMAR que treina um SVM com apenas 27 permissões.

Quando reduzimos o número de permissões de 113 (*baseline*) para 108 com PRNR, alcançamos taxas de precisão e acurácia mais altas, acima de 93%. Apesar de haver aumento no *recall*, o F1-Score e a taxa de falsos positivos (FPR) permaneceu mais baixa em relação a *baseline*, assim como o tempo de execução também diminuiu. Quando reduzimos o número de permissões de 113 para 30 com PRNR+SPR, mantiveram-se a acurácia e precisão na faixa de 90%, porém com F1-Score, a métrica a ser considerada, abaixo de 90%. Além disso, obtivemos um ganho de mais de 2 segundos em relação ao

tempo de execução.

Finalmente, quando aplicamos o último nível de seleção do SigPID chegamos a 27 permissões de um total de 113. Uma redução de 76% no número de permissões mantendo métricas como precisão e acurácia acima de 90% e atingindo uma redução no tempo de execução superior a 2 segundos. Porém, com a diminuição do número de permissões ocorre um aumento na taxa de falsos positivos.

Como o objetivo é chegar a um número mínimo de permissões que forneça uma acurácia acima de 90% e reduza o tempo de treino e teste dos modelos, podemos confirmar que os níveis de seleção de características do SigPID cumprem o seu papel.

Além do SVM, avaliamos também outros três algoritmos (*Random Forest*, *Decision Tree* e *Functional Trees* (GAMA, 2004)) e comparamos os resultados com os conjuntos de dados discriminados na Tabela 5.

Tabela 5 – Conjuntos de dados

Nº de Permissões	Conjunto de dados	Observação
113	*Baseline*	Contidas no *dataset* Drebin_215
22	SigPID	Identificadas no trabalho original do SigPID
32	Recorrentes	Identificadas por meio da interseção de permissões identificadas em outros trabalhos
22	Perigosas	Permissões na lista de perigosas da Google e estavam contidas na *baseline* deste trabalho
27	MLDP	Identificadas após aplicação dos três níveis de seleção sobre a *baseline*

A Tabela 6 sintetiza os resultados dos algoritmos *Random Forest*, *Decision Tree* e *Functional Trees*. As métricas apresentadas são a acurácia , que indica o desempenho geral do modelo, e F1-Score, a média harmônica entre o *recall* e a precisão.

Tabela 6 – Métricas de avaliação dos conjuntos de dados

	Decision tree		Random forest		Functional Trees	
Conjunto de dados	F1_Score	Acurácia	F1_Score	Acurácia	F1_Score	Acurácia
113 Baseline	92,28	94,32	94,44	95,94	97,30	97,27
22 SigPID	88,46	92,00	89,30	92,57	92,90	93,05
27 MLDP	91,09	93,50	92,62	94,64	95,40	95,43
32 Recorrentes	90,89	93,44	92,46	94,57	95,60	95,63
22 Perigosas Google	81,62	87,74	85,31	88,85	89,10	89,02

Quatro dos cinco conjuntos de dados obtiveram acurácia acima dos 90% (*Baseline*, SigPID, MLDP e Recorrentes). Destes, o *Baseline* performou melhor que os demais. Quando utilizamos o *Decision Tree* atingimos 94,32% de acurácia e 92,28% de F1-score.

Com os algoritmos *Random Forest* e *Functional Trees* atingimos 95,94% de acurácia e 94,44% de F1-score e acurácia e F1-score acima dos 97%, respectivamente.

Utilizando 22 permissões do SigPID a acurácia se manteve na faixa de 92% com *Decision Tree* e *Random Forest*, já F1-score se manteve abaixo de 90%. O *Functional Trees* foi o algoritmo com melhor desempenho, atingindo 93,05% acurácia e 92,90% de F1-score. Segundo os dados da Tabela 6, os conjuntos de dados MLDP e Recorrentes obtiveram resultados muito próximos, ficando ambos com acurácia acima de 95% utilizando o *Functional Trees*, por exemplo.

Na Figura 5 apresentamos os dados sobre o tempo de execução de cada algoritmo. Como podemos observar, o *Baseline* tem o maior tempo de execução, chegando a 11,31 segundos com algoritmo *Functional Trees*. Entretanto, é interessante observar que o problema ocorre apenas para o *Baseline*, já que o algoritmo executa em menos tempo que alguns dos demais para conjuntos de dados menores.

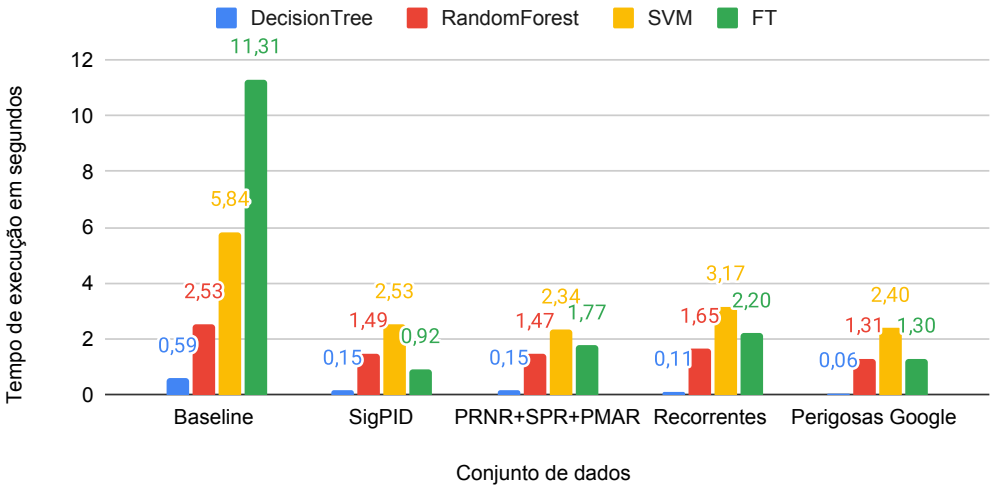

Figura 5 – Tempo de execução para diferentes conjuntos de dados

O *Functional Trees* é um algoritmo baseado em árvores de classificação que podem ter funções de regressão logística nos nós internos ou folhas. O algoritmo pode lidar com variáveis de destino binárias e multi-classe, atributos numéricos, nominais e valores ausentes, o que pode aumentar o tempo de execução de acordo com o número de características de entrada (GAMA, 2004). Como a estrutura do algoritmo é uma generalização das *Multivariate Trees* (GAMA, 2001), sua complexidade pode ser similar, isto é, $\mathcal{O}(n^2)$. Esta complexidade pode ser utilizada para explicar o comportamento apresentado no gráfico da Figura 5. De fato, o algoritmo *Functional Trees* reduziu significativamente o tempo de execução para conjuntos de dados menores (e.g., 1,77s para as 27 permissões do PRNR+SPR+PMAR).

Ao analisarmos os conjuntos de dados que utilizam 27 e 32 permissões, observa-se

que o tempo de execução e as métricas estão muito próximas para o *Functional Trees*. Nesse caso, podemos dizer que os modelos são equivalentes, isto é, não faz diferença utilizar as 32 permissões recorrentes ou as 27 permissões do MLDP. Isto indica que a identificação das permissões recorrentes, em trabalhos existentes na literatura, leva a resultados tão bom quanto os resultados do método de múltiplos níveis de seleção do SigPID, o que confirma uma das nossas hipóteses, isto é, permissões recorrentes podem ter um impacto positivo sobre os modelos de detecção de *malwares*.

5 API WEB E FERRAMENTA

5.1 Implementação

A implementação da ferramenta API Web tem dois objetivos: (a) testar se o modelo foi capaz de aprender com os dados de treino, ou seja, verificar como o modelo está se comportando com aplicativos jamais vistos; (b) coletar aplicativos que serão utilizados para criação de novos conjuntos de dados para aperfeiçoar o modelo.

Para salvar o modelo e coloca-lo em produção dividimos em três partes que consiste em: (a) serialização e desserialização do modelo; (b) descompilação do aplicativo; (c) disponibilização de acesso via API Web. A figura 6 representa a estrutura da API e as tecnologias utilizadas.

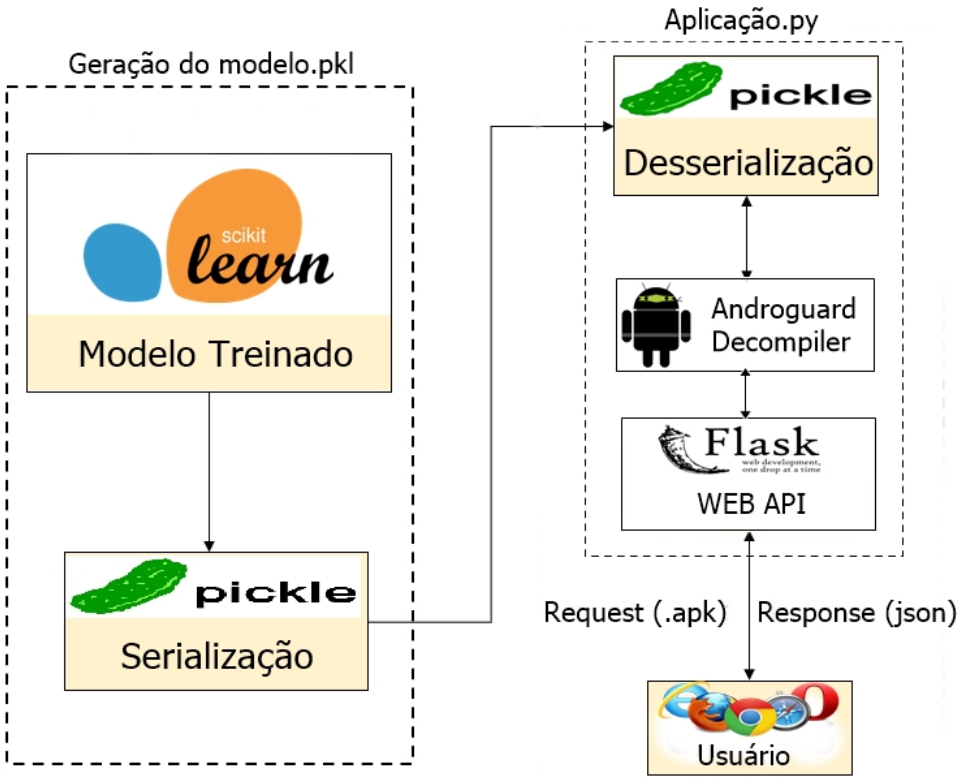

Figura 6 – Estrutura da API WEB

Após treinar e encontrar o melhor modelo, seria inviável treiná-lo toda vez que precisássemos executar a aplicação. Para evitar o retreino, utilizamos da biblioteca Pickle versão 3.10 para que os mesmos dados treinados possam ser usados posteriormente por

meio da desserialização.

O Flask (*micro-framework* voltado para o desenvolvimento web com Python) foi utilizado para a conceber a parte Web da plataforma. A interface é utilizada para que o usuário possa fazer o *upload* de um aplicativo no formato (APK) que passará por um processo de descompilação (*i.e.,* transformar o código objeto em código fonte inteligível) por meio de uma biblioteca chamada Androguard (versão 3.3.5).

Por meio do Androguard extraímos as permissões do aplicativo enviado pelo usuário e submetemos ao modelo preditivo desserializado, retornando o resultado da predição para o usuário no formato JSON (*JavaScript Object Notation*) contendo a classificação do aplicativo (*malware* ou benigno), nome do aplicativo, versão da API, versão miníma de API e uma lista de permissões declaradas no `manifest` do aplicativo analisado. A interface do processo de enviar um aplicativo para análise pode ser visto na figura 7, e a tela de resposta no formato JSON na figura 8.

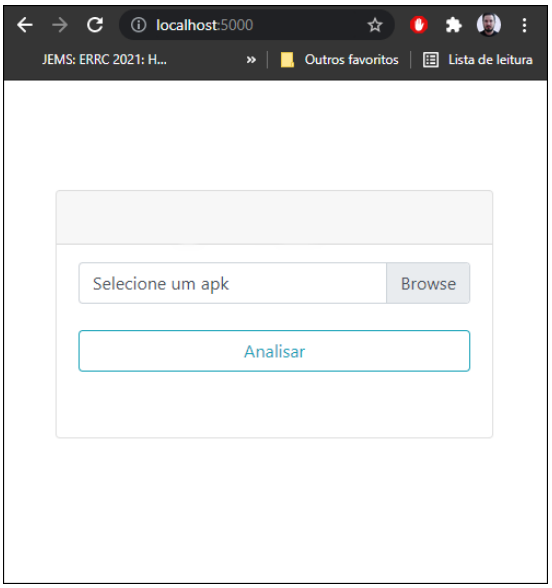

Figura 7 – Tela de upload

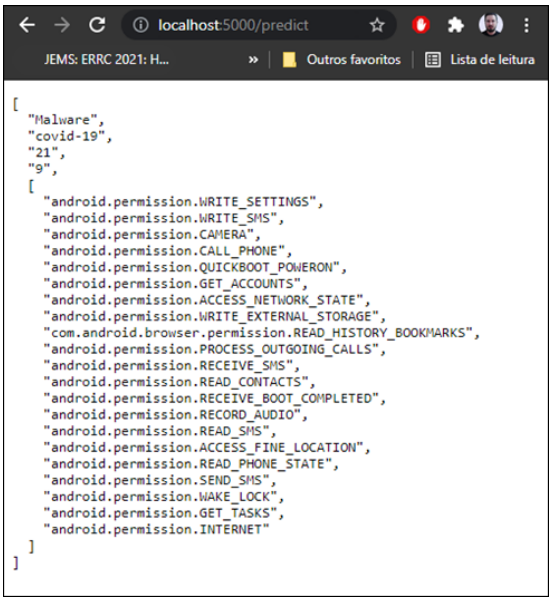

Figura 8 – Retorno da requisição

5.2 Avaliação

Para avaliar o desempenho da ferramenta selecionamos uma amostra qualitativa de 5 aplicativos benignos oriundos da PlayStore e 5 *malwares* oriundos do trabalho (WANG et al., 2021). A tabela 7 elenca os aplicativos selecionados; suas respectivas assinaturas criptográficas; nome dos aplicativos; versão de API e sua classificação, sendo "M"representando a classe de *malwares* e "B"a classe dos benignos.

Tabela 7 – Lista de aplicativos de teste

sha256sum	Nome	API	Class
c3bf7adf8403cee6ba7bdfa5a3fbf105a9071c21fa19d3c8d480cdf6f609d2a2	Istagram	30	B
9ef05ddc65957c7c867f855981c0229fcbaefde2b0b938db93631a0d0eb439a6	RealmCraft	30	B
aeda946f4099638ab800d19c13d94b66d1615d3a72d3388415476ae9eb2519e2	CartolaFC	29	B
e4986faf1c91f10574c252212bec22c8a51446ba30e2ec3b1705c0d25b635cd9	Spotify	29	B
aa754340578c312bb819c161fde74ea7b13e5a31f5fa3600deef4e1b40e94594	WhatsApp	29	B
87bde93b3bcc17cd4771898157dd9e8873277bc8fbd624a5e7799a41f6b2ffa1	Hudway	22	M
664b7f83235afcc5f08e6f55f74273a50a5cfddafbe185384c2cc4dfc6fc50b9	corona libya	22	M
17425e66428e284c2da73f3a7173e4291fb0b2bc76fd6d618921a9f0eb543340	covid-19	21	M
ff21b2149e5f4e915bbf3f94ddd12555f2a8dd8f5ec0a666410c6eb7c87513c4	Adobe Flash Player	22	M
ead4497c44414e025f15317e1755e809de24439bda3b4888d0686227dcd9d46b	AndroidSecureProduct	22	M

Infelizmente muitos conjuntos de dados são compostos de *malwares* antigos (*e.g.*, Android 2.1 - Eclair (API 7) e Android 4.4 - KitKat (API 19)) (SOARES et al., 2021). Por esse motivo selecionamos os *malwares* com versões de API mais atuais contidos no repositório escolhido. A versão mais atual encontrada foi Android 5.1 - Lollipop (API 22) que possui 9,20% de participação no mercado (BRADSHAW; ROUSSEL, 2020) conforme apresentado na figura 9.

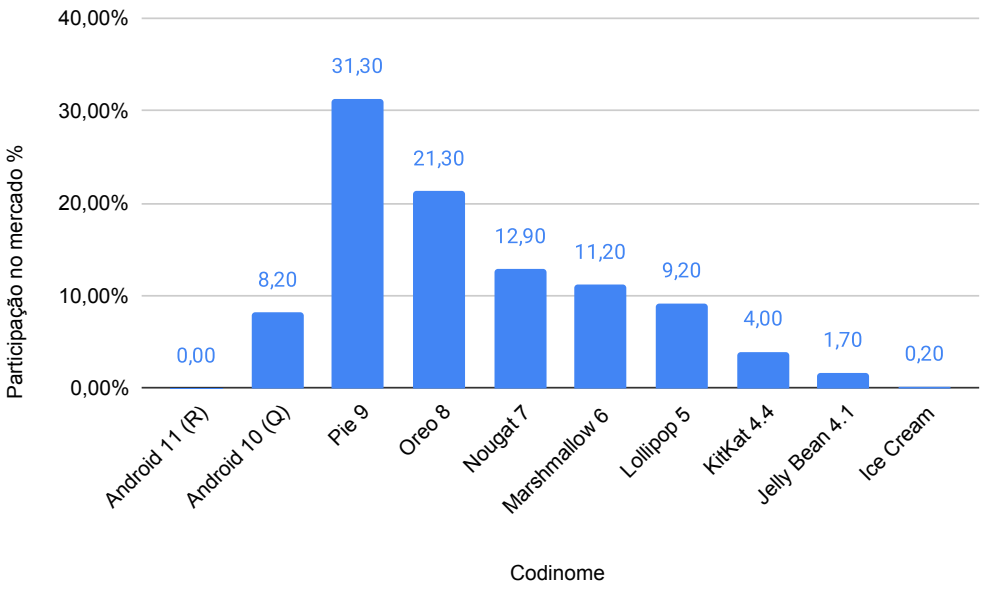

Figura 9 – Participação no mercado por versão do Android - (BRADSHAW; ROUSSEL, 2020)

Posteriormente comparamos os resultados da ferramenta com os resultados da plataforma VirusTotal[1] que é um serviço online gratuito que analisa arquivos e URLs,

[1] <https://www.virustotal.com/gui/home/upload>

possibilitando a identificação de conteúdo malicioso detectável por uma gama de antivírus e *scanners*. A tabela 8 apresenta os resultados de predição da API Web desenvolvida neste trabalho e da plataforma VirusTotal.

Tabela 8 – Comparativo de acurácia

Aplicativo	APIWeb	VirusTotal	VirusTotal Score
Istagram	✓	✓	0/63
RealmCraft	✓	✓	0/62
CartolaFC	✓	✓	0/63
Spotify	✓	✗	1/59
WhatsApp	✓	✓	0/59
Hudway	✗	✓	19/63
corona libya	✗	✓	19/63
covid-19	✓	✓	33/63
Adobe Flash Player	✓	✓	28/61
AndroidSecureProduct	✓	✓	10/62

Nossa ferramenta foi mais assertiva na classificação de aplicativos benignos quando comparada à plataforma VirusTotal (5 acertos contra 4 acertos da VirusTotal).

Quando testamos os aplicativos maliciosos, nossa ferramenta identificou corretamente 3 aplicativos (covid-19, AndroidSecureProduct e Adobe Flash Player), considerando 2 *malwares* como benignos (Hudway e corona libya). Neste ponto o VirusTotal levou vantagem, acertando os 5 aplicativos maliciosos. Entretanto nos aplicativos Hudway e corona libya – que nossa ferramenta falhou em classificar –, apenas 19 dos 63 *scanners* detectaram corretamente (*i.e.*, 30% dos scanners identificaram o *malware*). Para o aplicativo AndroidSecureProduct apenas 16% (10 de 62) *scanners* identificaram o *malware*, como podemos observar na tabela 8.

Se levarmos em consideração que a ferramenta utiliza apenas permissões para fazer a predição, podemos dizer que seu desempenho foi satisfatório. Um fator que possa ter impactado a capacidade de predição é o fato de que, embora o conjunto de dados utilizado (Drebin_215) seja do ano de 2018, seu conjunto de dados contém 5.560 aplicativos *malware* oriundos do trabalho (DANIEL et al., 2014), que por sua vez foram coletadas no período de agosto de 2010 a outubro de 2012, isso indica que a versão máxima dos aplicativos *malwares* é Android 4.1 - Jelly Bean (API 16).

6 DISCUSSÃO

O tempo de execução é o mais relevante que acurácia? Como pode ser observado nos dados apresentados, o algoritmo *Decision Tree* foi o que executou no menor tempo em todos os conjuntos de dados, ficando abaixo de 1 segundo. Entretanto, o algoritmo ficou abaixo do *Decision Tree* e *Functional Tree* em relação às métricas de desempenho acurácia e F1-Score. Portanto, cabe ao usuário final realizar uma análise do *trade-off* entre tempo de execução e desempenho, isto é, latência do aprendizado *versus* capacidade de detecção do modelo.

O tempo de execução pode tornar-se irrelevante. Como executamos os experimentos em um ambiente com baixa capacidade computacional, acreditamos que o tempo de execução seja praticamente irrelevante quando aplicarmos os modelos em *smartphones* atuais. Enquanto que os experimentos foram realizados utilizando uma CPU Intel Celeron 1007U de 1.5GHz, que produz apenas 96 Gigaflops, *smartphones* modernos disponibilizam CPUs como a Qualcomm Snapdragon 865, que opera em 2.84GHz e é capaz de produzir 1.228 Gigaflops, ou seja, um poder computacional mais de 12x maior. Outros fatores que irão impactar o tempo de execução são as instruções avançadas em CPUs modernas, voltadas para aprendizado de máquina, e a velocidade da memória RAM. Enquanto que o experimento foi executado em um hardware com 4GB de RAM operando a 1.600MHz, um *smartphone* moderno fornece 6GB de RAM operando a 2.750MHz.

Quantidade de dados impacta o tempo de execução. Conjuntos de dados maiores e mais atuais impactam o desempenho dos algoritmos de aprendizado de máquina. Por exemplo, o algoritmo SVM apresenta problemas de desempenho para conjuntos de dados maiores, aumentando substancialmente o tempo treinamento (CRISTIANINI; SHAWE-TAYLOR et al., 2000). Algo similar pode ser dito do algoritmo *Functional Trees*, que apresentou um tempo de computação substancialmente maior para 113 permissões (i.e., um conjunto com mais características), por exemplo.

Desafios. Encontramos diversos problemas de reprodutibilidade do SigPID, como a falta de informação sobre os hiper-parâmetros utilizados nos algoritmos, indisponibilidade de conjuntos de dados e falta de detalhamento de ferramentas e tecnologias utilizadas no SigPID.

7 CONSIDERAÇÕES FINAIS

Aplicando os 3 níveis de seleção do SigPID nas 113 permissões do *dataset* Drebin_215, conseguimos reduzir em 76% o número de permissões a serem analisadas para detecção de *malwares* Android, mantendo acurácia acima de 90% com SVM, 93,50% com *Decision Tree*, 94,64% com *Random Forest* e 95% com o *Functional Trees*. Além disso, conseguimos também reduzir o tempo de execução dos modelos, porém, ao custo de um leve aumento na taxa de falsos positivos.

Além das 27 permissões resultantes dos 3 níveis de seleção, utilizamos também conjuntos de 113, 22 e 32 permissões. Percebemos que o conjunto de dados que utiliza todas permissões (113) foi o que melhor performou (e.g., 97% de acurácia), porém, ao preço de de um tempo de execução significativamente maior. Um caso interessante foi quando comparamos os conjuntos com 22 permissões, sendo um oriundo do trabalho original do SigPID e 22 permissões classificadas como perigosas pela Google. O SigPID chegou a 93% de acurácia enquanto que as perigosas manteve acurácia abaixo de 90%, o que indica que o fato da permissão ser classificada como perigosa não a torna necessariamente relevante para detecção de *malwares*. Outra observação interessante é o fato de os conjuntos de dados com 32 permissões mais recorrentes e as 27 identificadas aplicando os 3 níveis de seleção atingiram resultados muito próximos. Isto indica que escolher as permissões de acordo com a recorrência pode ser um caminho de investigação a ser seguido.

Como trabalhos futuros, podemos elencar:

1. testes com conjuntos de dados maiores;
2. testes com conjuntos de dados atuais;
3. avaliação dos níveis de seleção para outras características (e.g., *intents* e chamadas de API);
4. otimização de hiperparâmetros;
5. testar os modelos em *smartphones* modernos;
6. mensurar o tempo de execução dos modelos em CPUs modernas projetadas para acelerar a computação de algoritmos de aprendizado de máquina;
7. criar um *dataset* com dados atualizados utilizando os recursos da ferramenta proposta, bem como *scanners* como o VirusTotal;
8. melhorar a ferramenta (interface e segurança).

REFERÊNCIAS

AGRAWAL, R.; SRIKANT, R. et al. Fast algorithms for mining association rules. In: CITESEER. **Proc. 20th int. conf. very large data bases, VLDB**. [S.l.], 1994. v. 1215, p. 487–499. Citado na página 25.

ALSOGHYER, S.; ALMOMANI, I. On the effectiveness of application permissions for android ransomware detection. In: **2020 6th Conference on Data Science and Machine Learning Applications (CDMA)**. [S.l.: s.n.], 2020. p. 94–99. Citado na página 17.

ARP, D. et al. Drebin: Effective and explainable detection of android malware in your pocket. In: **Ndss**. [S.l.: s.n.], 2014. v. 14, p. 23–26. Citado na página 19.

ASSOLIN, J. et al. Permissões android para detecção de malwares: Um estudo preliminar. In: **XV Workshop de Trabalhos de Iniciação Científica e de Graduação (WTICG)**. [S.l.: s.n.], 2021. Citado 3 vezes nas páginas 17, 18 e 27.

BAYAZIT, E. C.; SAHINGOZ, O. K.; DOGAN, B. Malware detection in android systems with traditional machine learning models: a survey. In: IEEE. **2020 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)**. [S.l.], 2020. p. 1–8. Citado na página 17.

BOUTABA, R. et al. A comprehensive survey on machine learning for networking: evolution, applications and research opportunities. **Journal of Internet Services and Applications**, v. 9, n. 1, p. 16, Jun 2018. ISSN 1869-0238. Disponível em: <https://doi.org/10.1186/s13174-018-0087-2>. Citado na página 23.

BRADSHAW, K.; ROUSSEL, D. **Distribution data for Android**. 2020. <https://androiddistribution.io/>. Citado na página 33.

CHAKKARAVARTHY, S. S.; SANGEETHA, D.; VAIDEHI, V. A survey on malware analysis and mitigation techniques. **Computer Science Review**, Elsevier, v. 32, p. 1–23, 2019. Citado na página 17.

CRISTIANINI, N.; SHAWE-TAYLOR, J. et al. **An introduction to support vector machines and other kernel-based learning methods**. [S.l.]: Cambridge university press, 2000. Citado na página 35.

DANIEL, A. et al. Drebin: Efficient and explainable detection of android malware in your pocket". In: **Proceedings of 21th Annual Network and Distributed System Security Symposium (NDSS)**. [S.l.: s.n.], 2014. Citado na página 34.

GAMA, J. Functional trees for classification. In: IEEE. **Proceedings 2001 IEEE International Conference on Data Mining**. [S.l.], 2001. p. 147–154. Citado na página 29.

GAMA, J. Functional trees. **Machine learning**, Springer, v. 55, n. 3, p. 219–250, 2004. Citado 2 vezes nas páginas 28 e 29.

Google Developer. **Request app permissions**. 2021. <https://developer.android.com/training/permissions/requesting>. Online; accessed 8 Jun 2021. Citado na página 17.

GYAMFI, N. K.; OWUSU, E. Survey of mobile malware analysis, detection techniques and tool. In: IEEE. **2018 IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)**. [S.l.], 2018. p. 1101–1107. Citado na página 17.

JAMES, G. et al. **An introduction to statistical learning**. [S.l.]: Springer, 2013. v. 112. Citado na página 19.

Li, J. et al. Significant permission identification for machine-learning-based android malware detection. **IEEE Transactions on Industrial Informatics**, v. 14, n. 7, p. 3216–3225, 2018. Citado 3 vezes nas páginas 17, 21 e 27.

SHARMA, T.; RATTAN, D. Malicious application detection in android—a systematic literature review. **Computer Science Review**, Elsevier, v. 40, p. 100373, 2021. Citado na página 17.

SOARES, T. et al. Detecção de malwares android: datasets e reprodutibilidade. In: **XV Workshop de Trabalhos de Iniciação Científica e de Graduação (WTICG)**. [S.l.: s.n.], 2021. Submetido - em revisão. Citado na página 33.

SUN, L. et al. Sigpid: significant permission identification for android malware detection. In: IEEE. **2016 11th international conference on malicious and unwanted software (MALWARE)**. [S.l.], 2016. p. 1–8. Citado na página 17.

WANG, L. et al. **Beyond the Virus: A First Look at Coronavirus-themed Mobile Malware**. 2021. Citado na página 32.

WU, Q.; ZHU, X.; LIU, B. A survey of android malware static detection technology based on machine learning. **Mobile Information Systems**, Hindawi, v. 2021, 2021. Citado na página 17.

YERIMA, S. Y.; SEZER, S. **Droidfusion: A novel multilevel classifier fusion approach for android malware detection**. 2018. <https://figshare.com/articles/dataset/Android_malware_dataset_for_machine_learning_2/5854653>. Citado na página 19.

YILDIZ, O.; DOĞRU, I. A. Permission-based android malware detection system using feature selection with genetic algorithm. **International Journal of Software Engineering and Knowledge Engineering**, World Scientific, v. 29, n. 02, p. 245–262, 2019. Citado na página 17.

Apêndices

APÊNDICE A – PERMISSÕES ANDROID PARA DETECÇÃO DE MALWARES: UM ESTUDO PRELIMINAR

Permissões Android para Detecção de Malwares: Um Estudo Preliminar

Joner Assolin, Guilherme Siqueira, Gustavo Rodrigues, Diego Kreutz

[1] Universidade Federal do Pampa (Unipampa)
{NomeSobrenome}.aluno@unipampa.edu.br,kreutz@unipampa.edu.br

Resumo. *Além da quantidade de aplicativos benignos e malignos, outro fator que dificulta a detecção de malwares Android é o grande número de características para análise estática ou dinâmica utilizando métodos de aprendizagem de máquina. Como forma de atacar o desafio de escalabilidade derivado deste contexto, há trabalhos que propõem a utilização de um número reduzido de permissões, como é o caso do SigPID. Neste trabalho, apresentamos um passo inicial na realização do (a) mapeamento das permissões mais recorrentes em trabalhos existentes; (b) mapeamento dos requisitos para a reprodução do SigPID; e (c) implementação e avaliação dos métodos de aprendizagem do SigPID, utilizando um dataset publicamente disponível. Nós comparamos o trabalho original do SigPID, que utiliza 22 permissões, com as 32 permissões identificadas como mais recorrentes; as 113 permissões do dataset público escolhido; e as 22 permissões (contidas no dataset) consideradas perigosas pela Google. Nosso estudo inicial indica que o número de permissões impacta o tempo de treinamento e execução, bem como a acurácia dos modelos. Entretanto, o tempo de execução pode não ser significativo a ponto de justificar um número menor de permissões para detecção de malwares em tempo de instalação do APK (*e.g.*, no próprio smartphone do usuário final).*

1. Introdução

Podemos encontrar diversos trabalhos na literatura que propõem técnicas baseadas em aprendizado de máquina para detectar *malwares* em sistemas operacionais Android [Bayazit et al., 2020, Wu et al., 2021]. Esses trabalhos utilizam abordagens de análise estática, dinâmica ou híbrida [Sharma and Rattan, 2021, Gyamfi and Owusu, 2018]. Independentemente da abordagem, utilizar permissões permite um bom desempenho na detecção de *malwares* em Android [Alsoghyer and Almomani, 2020]. Entretanto, selecionar quais utilizá-las ainda é um desafio uma vez que existe um grande número de permissões disponíveis no sistema Android, e dependendo da escolha, pode impactar negativamente o tempo de execução das soluções existentes. Em soluções baseadas em aprendizado de máquina, o número de características (*e.g.*, permissões) utilizadas impacta no tempo de treino e também na acurácia dos modelos [Chakkaravarthy et al., 2019]. Com o objetivo de mitigar este problema, há estudos que investigam o impacto da redução do números de permissões utilizadas para treino dos modelos no tempo de execução. Alguns estudos verificaram que, mesmo utilizando um número menor de permissões, a acurácia do modelo não é comprometida mas o tempo de execução é reduzido (*i.e.*, e há aumento da escalabilidade sem comprometer o desempenho da classificação) [Li et al., 2018, Yildiz and Doğru, 2019].

O objetivo deste trabalho pode ser dividido em três partes. Primeiro, realizar um estudo para identificar as permissões mais recorrentes em trabalhos de detecção de *malwares* utilizando aprendizado de máquina. Segundo, reproduzir o trabalho denominado SigPID [Li et al., 2018][1], que é uma evolução dos trabalhos [Sun et al., 2016, Wang et al., 2014], cujo foco é oferecer escalabilidade para o processo de detecção de *malwares* reduzindo o número total de permissões utilizadas. Finalmente, numa terceira etapa, avaliar o desempenho do SigPID comparado com versões dos modelos adaptados para as permissões mais recorrentes (identificadas na primeira etapa) e para o conjunto de permissões consideradas como perigosas pela Google.

Nesta primeira fase do trabalho, realizamos: (a) o mapeamento das permissões mais recorrentes; (b) o mapeamento dos requisitos para a reprodução do SigPID; (c) a implementação dos múltiplos níveis de poda de dados multi-nível do SigPID para extrair as permissões mais significativas; e (d) uma discussão dos resultados dos algoritmos *Random Forest*, *Decision Tree* e *Support Vector Machine* (SVM).

O trabalho está organizado como segue. Na Seção 2, é apresentada uma discussão sobre permissões recorrentes em *malwares*. A Seção 3 apresenta uma visão geral sobre a reprodutibilidade do SigPID. As Seções 4 e 5 apresentam uma descrição da estratégia adotada para execução do trabalho e os detalhes técnicos de implementação, respectivamente. Na Seção 6 apresentamos os primeiros resultados do trabalho. Por fim, na Seção 7 apresentamos uma discussão sobre descobertas realizadas.

2. Permissões Recorrentes

Na primeira etapa do levantamento das permissões recorrentes em aplicativos malignos, utilizamos nove trabalhos que utilizam análise estática ou dinâmica de permissões para detecção de *malwares* [Peiravian and Zhu, 2013, Lopez and Cadavid, 2016, Alsoghyer and Almomani, 2020, Idrees et al., 2017, Sangal and Verma, 2020, Tam et al., 2017, Li et al., 2018, Martín et al., 2019, AYsan and sen, 2015]. A Figura 1 apresenta as 32 permissões mais comuns (pelo critério de ocorrência em dois ou mais trabalhos analisados) de um total de 41 permissões identificadas. Das 32 permissões, 11 são abordadas em mais de 50% dos trabalhos. Destas 11, 6 são consideradas perigosas pela Google (READ_PHONE_STATE, SEND_SMS, ACCESS_COARSE_LOCATION, WRITE_EXTERNAL_STORAGE, ACCESS_FINE_LOCATION e READ_SMS).

A partir da API 23 do Android, de agosto de 2015, as permissões perigosas são solicitadas em tempo de execução. Em versões anteriores da API, todas as permissões eram concedidas durante a instalação do aplicativo. Das permissões mais recorrentes, consideradas como normais, INTERNET, RECEIVE_BOOT_COMPLETED, ACCESS_WIFI_STATE, ACCESS_NETWORK_STATE e WAKE_LOCK são autorizadas durante a instalação do aplicativo.

Permissões como ACCESS_NETWORK_STATE, ACCESS_WIFI_STAT, KILL_BACKGROUND_PROCESSES, RECEIVE_BOOT_COMPLETED, INTERNET, WAKE_LOCK e CHANGE_NETWORK_STATE, apesar de não serem consideradas perigosas, aparecem com frequência em *malwares*. Isto acontece

[1]O SigPID é um dos principais trabalhos de escalabilidade no processo de detecção de *malwares* Android utilizando permissões, a característica mais utilizada para este fim.

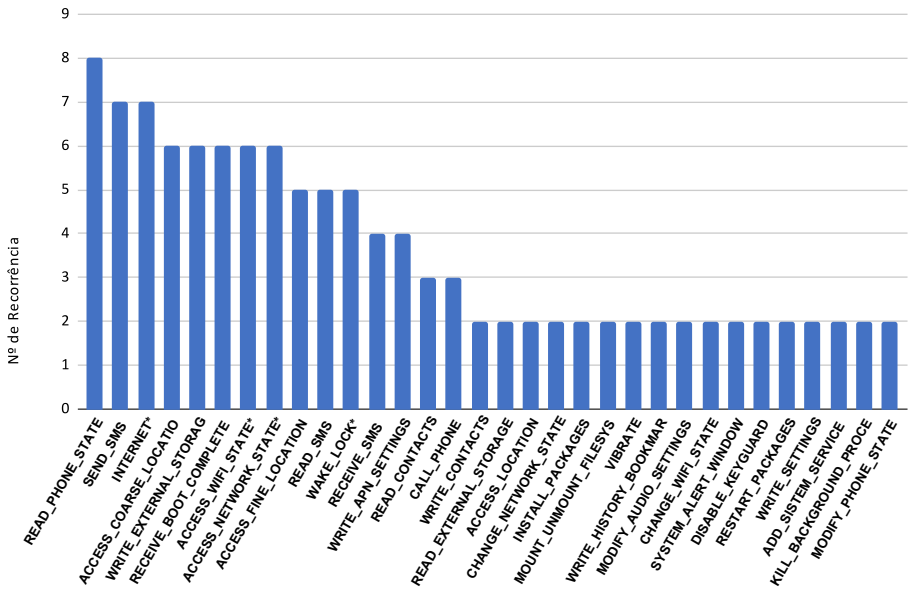

Figura 1. Permissões recorrentes

por que a maioria dos aplicativos malignos e benignos necessitam de acesso a rede [Sangal and Verma, 2020, Tam et al., 2017, Alsoghyer and Almomani, 2020]

As permissões de gravação em armazenamento externo (*e.g.,* WRITE_EXTERNAL_STORAGE) e acesso ao estado do smartphone (*e.g.*, READ_PHONE_STATE - para acesso a dados como IMEI, número de telefone) são frequentemente solicitadas em aplicativos maliciosos e benignos. Os aplicativos maliciosos tendem a solicitar com mais frequência as permissões relacionadas a SMS, ACCESS_COARSE_LOCATION, ACCESS_WIFI_STATE. O acesso ao SMS pode indicar relação com ataques que visam interceptar, gerar ou vazar mensagens do usuário (*e.g.*, códigos de verificação, *tokens*, senhas e outros) [Peiravian and Zhu, 2013, Idrees et al., 2017, AYsan and sen, 2015].

3. Reprodutibilidade do SigPID

Para avaliar o desempenho do SigPID quando comparado com versões dos modelos adaptados para as permissões mais recorrentes e para o conjunto de permissões consideradas como perigosas pela Google, o primeiro passo do trabalho foi reproduzir o trabalho original. O SigPID identifica e utiliza 22 permissões (de 135 extraídas dos APKs), consideradas as mais significantes, e algoritmos de aprendizado supervisionado para detectar *malwares*.

No trabalho do SigPID, os autores informam que utilizaram uma amostra de 310.926 aplicativos benignos da Google Play e 5.494 *malwares*, cujas origens são os *datasets* Mal Zhou (1.260), Mal Com1 (247), Mal Com2 (154), Mal VS (3.207) [Li et al., 2018, Wang et al., 2014]. Porém, o processo de acesso e utilização destes *datasets* não é detalhado, o que impossibilita a replicação do *dataset* original do trabalho. Infelizmente, este é um cenário muito comum em trabalhos de detecção de *malwares* Android [Soares et al., 2021].

Além dos problemas relacionados ao *dataset*, também não há detalhes sobre as tecnologias utilizadas (como *frameworks* e bibliotecas) e sobre os hiper-parâmetros utilizados. Sem os hiper-parâmetros utilizados nos modelos, que são variáveis de configuração que controlam o processo de treinamento, é tecnicamente inviável de reproduzir e validar os resultados. Em síntese, a presença desses problemas são fatores que impossibilitam a reprodução fidedigna do estudo.

4. Estratégia Adotada

A primeira estratégia cogitada foi entrar em contato com os autores do SigPID para obter acesso aos *datasets* e as configurações importantes dos modelos, como os hiper-parâmetros. Entretanto, o acesso a esses dados depende de uma resposta positiva dos autores do SigPID. Paralelamente, traçamos uma estratégia alternativa, descrita a seguir.

Como no momento não podemos reproduzir fidedignamente o SigPID, buscamos um *dataset* que, assim como no SigPID, disponibiliza permissões de aplicativos malignos ou benignos como *features*. O *dataset* Drebin_215[2], disponível publicamente no FigShare (`https://figshare.com`), site especializado em disponibilizar dados de pesquisas, possui 215 atributos extraídos de 15.036 aplicativos (5.560 malignos e 9.476 benignos).

Para o nosso trabalho, utilizamos apenas os recursos de permissões do Android; outros recursos como chamadas de APIs e *Intents* não foram considerados. Identificamos um total de 113 permissões no *dataset* Drebin_215. Destas permissões (*Baseline* 113), derivamos três *subsets*: (a) SigPID 22: as mesmas 22 permissões utilizadas pelo SigPID; (b) Perigosas 22: 22 permissões consideradas perigosas pela Google; e (c) Recorrentes 32: as 32 permissões recorrentes identificadas na Seção 2. Cada *subset* conta com uma amostra de 15.036 aplicativos (5.560 malignos e 9.476 benignos).

5. Implementação

Na primeira etapa, foram utilizados os algoritmos *Random Forest*, *Decision Tree* e SVM, todos também utilizados pelo SigPID. Para implementação dos modelos de aprendizado de máquina, foi utilizada a biblioteca Scikit-learn versão 0.22.1, através da ferramenta Jupyter Notebook (IPython 7.12.0, Python 3.7.6 (default, Jan 8 2020) com o navegador *Google Chrome* Versão 91.0.4472.124 (Versão oficial) 64 bits. Para execução foi utilizado um notebook com processador Intel Celeron 1007U (1.5GHz, Dual Core, 2MB L2), 4GB DDR3 1.600MHz, disco rígido de 320GB (SATA - 5.400rpm), Windows 10 Home Single Language, compilação 19042.1110.

Para garantir a reprodutibilidade do experimento, definimos arbitrariamente *random_state* 1 para embaralhar o conjunto de dados inicial. Utilizamos uma divisão pseudo-aleatória (*random split*) de 70%/30%, a partir dos dados iniciais, sendo 70% utilizado para treinos e 30% para testes [James et al., 2013]. Também utilizamos divisões estratificadas (*stratify*), pois são desejáveis em casos de conjunto de dados desbalanceados. Os hiper-parâmetros foram definidos como padrão referente a versão 0.22.1 da biblioteca *Scikit-learn* que define o número de árvores na floresta (`n_estimators=100`) e a profundidade máxima da árvore (`max_depth=None`) para o RandomForestClassifier, por exemplo. A implementação dos algoritmos, incluindo a definição detalhada dos

[2]`https://figshare.com/articles/dataset/Android_malware_dataset_for_machine_learning_2/5854653`

hiper-parâmetros, está disponível em `https://github.com/Malware-Hunter/SigPID`.

6. Resultados Preliminares

Para entender quão bom um modelo preditivo é, existem métricas para avaliar o desempenho (ou generalização) de um método de aprendizagem de máquina [Amidi and Amidi, 2020]. As métricas na comparação inicial dos modelos com os diferentes *subsets* de características (ver Seção 4) são: **Acurácia**: mede a frequência com que o modelo consegue acertar predições; **Precisão**: considerando as predições para classe positiva, mede a frequência com que tais predições foram feitas corretamente; **Recall**: considerando os valores reais, mede a proporção dos positivos que foram corretamente previstos; e **F1-Score** é uma média harmônica entre a precisão e o *recall*.

A Tabela 1 representa as métricas de avaliação do algoritmo *Random Forest* sobre os quatro *subsets* de permissões. As métricas estão divididas em duas classes (B) e (M), onde (B) representa o conjunto de aplicativos benignos e (M) o conjunto de aplicativos malignos.

Tabela 1. Métricas de avaliação *Random Forests*

RandomForest					
Nº de permissões	**Classe**	**precisão**	**recall**	**f1-score**	**acurácia**
Baseline 113	B	96	98	97	96
	M	96	93	94	
SigPID 22	B	91	98	94	92
	M	95	83	89	
Perigossas 22	B	86	96	91	88
	M	92	74	82	
Recorrentes 32	B	94	97	96	95
	M	95	90	93	

Podemos observar que atingimos melhores resultados, em todas as métricas em ambas as classes, quando utilizamos 113 permissões. Entretanto, a Tabela 2 ilustra que há uma penalidade de custo (em segundos) em relação ao tempo de execução de treino e teste (*i.e.,* o dobro do tempo dos demais casos).

Tabela 2. Tempo de execução dos algoritmos por número de permissões

	Subset de Permissões			
Algoritmos	Baseline 113	SigPID 22	Perigosas 22	Recorrentes 32
RandomForest	2.531	1.288	1.239	1.589
DecisionTree	0.236	0.076	0.064	0.101
SVM	5.564	2.742	2.414	3.499

Quando reduzimos o número de permissões para as 22 catalogadas pelo SigPID, notamos que obtivemos o mesmo valor de *Recall* (98%) para a classe B. Também podemos notar que, mesmo com uma redução significativa no número de permissões, é possível atingir uma acurácia acima dos 90% e reduzir o tempo de execução pela metade.

Considerando as 22 permissões perigosas da Google, notamos um baixo desempenho de detecção em ambas as classes e em todas as métricas, resultando em uma acurácia

abaixo de 90%. Porém, quando utilizamos o conjunto de 32 permissões recorrentes, notamos que as métricas tiveram resultado melhor e, na verdade, muito próximo às 113 da *baseline*, atingindo 95% de acurácia. Além disso, o tempo de execução ficou próximo ao obtido pelo SigPID 22. Em outras palavras, temos um bom desempenho de detecção com um baixo tempo de execução.

O *Decision Tree* teve uma queda de 2% na acurácia para 113 permissões do *Random Forests*. Entretanto, teve um tempo de execução significativamente menor, conforme detalhado na Tabela 2. Com 32 permissões, a perda de acurácia foi de 1%, acompanhado de um ganho significativo no tempo de execução.

Tabela 3. Métricas de avaliação *Decision Tree*

DecisionTree					
Nº de permissões	**Classe**	**precision**	**recall**	**f1-score**	**Acurácia**
Baseline 113	B	95	96	96	94
	M	93	92	92	
SigPID 22	B	91	97	94	92
	M	95	83	88	
Perigossas 22	B	86	96	91	88
	M	92	74	82	
Recorrentes 32	B	93	97	95	94
	M	94	88	91	

O SVM foi o modelo que apresentou o pior desempenho de detecção e de tempo de execução. Tomando como exemplo o SigPID 22, o SVM obteve uma acurácia de 88% e resultou em um tempo de execução significativamente maior que os demais algoritmos.

Tabela 4. Métricas de avaliação *SVM*

SVM					
Nº de permissões	Classe	precision	recall	f1-score	Acurácia
Baseline 113	B	93	96	95	93
	M	93	88	91	
SigPID 22	B	87	96	91	88
	M	92	75	82	
Perigossas 22	B	90	88	89	86
	M	80	84	82	
Recorrentes 32	B	90	94	92	89
	M	89	82	85	

7. Discussão

Os resultados preliminares indicam que utilizar as 113 permissões leva a um melhor desempenho de detecção, porém com um maior tempo de execução. Dependendo do cenário (*e.g.*, detecção de *malware* no *smartphone* do usuário final), o tempo de detecção (*e.g.*, 1s ou 2s) pode não ser um problema. Eventualmente, um melhor desempenho de detecção é mais relevante que um menor tempo de execução. Vale ressaltar que os testes foram realizados em uma máquina obsoleta (ver Seção 5). Esses tempos (*e.g.*, 2,5s para o *Baseline* 113) irão reduzir significativamente com *smartphones* modernos, que contam com processadores de 2.4 GHz e 8 cores, por exemplo.

O desempenho dos algoritmos *Random Forest* e *Decision Tree* pode ser considerado muito bom com as 32 permissões recorrentes. Isto é um indicativo de que as permissões recorrentes podem ser um caminho interessante na busca por desempenho de detecção e baixo tempo de execução.

Os nossos resultados indicam que permissões consideradas perigosas pela Google não possuem uma relevância suficientemente significativa em modelos preditivos. O *subset* SigPID 22 atingiu um desempenho de detecção superior às Perigosas 22.

Recomendações

Os estudos de detecção de *malwares*, baseados em modelos de aprendizagem de máquina, deveriam disponibilizar informações detalhadas sobre os *datasets* utilizados. Além disso, e preferencialmente, utilizar *datasets* publicamente disponíveis, permitindo assim a validação e reprodução do trabalho.

Com relação aos modelos preditivos, é importante os trabalhos fornecerem informações relevantes sobre os hiper-parâmetros utilizados nos algoritmos de aprendizagem de máquina. Além disso, os trabalhos devem fornecer também informações detalhadas sobre as ferramentas e tecnologias utilizadas, incluindo bibliotecas e suas respectivas versões, por exemplo. Esses dados são cruciais para a reprodutibilidade do trabalho.

Trabalhos futuros

Como trabalhos futuros, podemos elencar:

- Aumentar o número de trabalhos mapeados para identificar as permissões recorrentes em detecção de *malwares* Android;
- Investigar o impacto das novas APIs nos métodos de detecção de *malwares* baseados em permissões;
- Criar um novo *dataset* a partir da extração de permissões de um conjunto grande de APKs maliciosos atuais (*i.e.*, que utilizam as novas APIs do Android);
- Otimizar os hiper-parâmetros dos modelos preditivos utilizados neste trabalho.

Referências

Alsoghyer, S. and Almomani, I. (2020). On the effectiveness of application permissions for android ransomware detection. In *2020 6th Conference on Data Science and Machine Learning Applications (CDMA)*, pages 94–99.

Amidi, A. and Amidi, S. (2020). Machine learning tips and tricks cheatsheet.

AYsan, A. I. and sen, S. (2015). Api call and permission based mobile malware detection (in english). In *2015 23nd Signal Processing and Communications Applications Conference (SIU)*, pages 2400–2403.

Bayazit, E. C., Sahingoz, O. K., and Dogan, B. (2020). Malware detection in android systems with traditional machine learning models: a survey. In *2020 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)*, pages 1–8. IEEE.

Chakkaravarthy, S. S., Sangeetha, D., and Vaidehi, V. (2019). A survey on malware analysis and mitigation techniques. *Computer Science Review*, 32:1–23.

Gyamfi, N. K. and Owusu, E. (2018). Survey of mobile malware analysis, detection techniques and tool. In *2018 IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, pages 1101–1107. IEEE.

Idrees, F., Rajarajan, M., Conti, M., Chen, T. M., and Rahulamathavan, Y. (2017). Pindroid: A novel android malware detection system using ensemble learning methods. *Computers & Security*, 68:36–46.

James, G., Witten, D., Hastie, T., and Tibshirani, R. (2013). *An introduction to statistical learning*, volume 112. Springer.

Li, J., Sun, L., Yan, Q., Li, Z., Srisa-an, W., and Ye, H. (2018). Significant permission identification for machine-learning-based android malware detection. *IEEE Transactions on Industrial Informatics*, 14(7):3216–3225.

Lopez, C. C. U. and Cadavid, A. N. (2016). Machine learning classifiers for android malware analysis. In *2016 IEEE Colombian Conference on Communications and Computing (COLCOM)*, pages 1–6.

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.

Peiravian, N. and Zhu, X. (2013). Machine learning for android malware detection using permission and api calls. In *2013 IEEE 25th international conference on tools with artificial intelligence*, pages 300–305. IEEE.

Sangal, A. and Verma, H. K. (2020). A static feature selection-based android malware detection using machine learning techniques. In *2020 International Conference on Smart Electronics and Communication (ICOSEC)*, pages 48–51.

Sharma, T. and Rattan, D. (2021). Malicious application detection in android—a systematic literature review. *Computer Science Review*, 40:100373.

Soares, T., Siqueira, G., Barcellos, L., Sayyed, R., Vargas, L., Rodrigues, G., Assolin, J., Pontes, J., and Kreutz, D. (2021). Detecção de malwares android: datasets e reprodutibilidade. `https://arxiv.kreutz.xyz/mh21_reprodutibilidade.pdf`.

Sun, L., Li, Z., Yan, Q., Srisa-an, W., and Pan, Y. (2016). Sigpid: significant permission identification for android malware detection. In *2016 11th international conference on malicious and unwanted software (MALWARE)*, pages 1–8. IEEE.

Tam, K., Feizollah, A., Anuar, N. B., Salleh, R., and Cavallaro, L. (2017). The evolution of android malware and android analysis techniques. *ACM Computing Surveys (CSUR)*, 49(4):1–41.

Wang, W., Wang, X., Feng, D., Liu, J., Han, Z., and Zhang, X. (2014). Exploring permission-induced risk in android applications for malicious application detection. *IEEE Transactions on Information Forensics and Security*, 9(11):1869–1882.

Wu, Q., Zhu, X., and Liu, B. (2021). A survey of android malware static detection technology based on machine learning. *Mobile Information Systems*, 2021.

Yildiz, O. and Doğru, I. A. (2019). Permission-based android malware detection system using feature selection with genetic algorithm. *International Journal of Software Engineering and Knowledge Engineering*, 29(02):245–262.

APÊNDICE B – DETECÇÃO DE MALWARES ANDROID: REPRODUÇÃO DOS NÍVEIS DE CORTE DO SIGPID

Detecção de Malwares Android: reprodução da seleção de características do SigPID

Joner Assolin[1], Vanderson Rocha[2], Guilherme Silveira[1], Gustavo Rodrigues[1], Eduardo Feitosa[2], Karina Casola[1], Diego Kreutz[1]

[1] Universidade Federal do Pampa (Unipampa)
[2]Universidade Federal do Amazonas (UFAM)

Resumo. *Para atacar o desafio de escalabilidade na detecção de malwares Android, há trabalhos que propõem a utilização de um número reduzido de permissões, como é o caso do SigPID. Neste trabalho, apresentamos a reprodução dos 3 níveis de seleção de permissões e avaliação dos principais métodos de aprendizagem do SigPID, utilizando um conjunto de dados publicamente disponível. Nosso estudo inicial indica que o número de permissões impacta o tempo de treinamento e execução, bem como a acurácia dos modelos. Entretanto, o tempo de execução pode não ser significativo a ponto de justificar um número menor de permissões para detecção de malwares.*

1. Introdução

Dentre os métodos para detecção de *malwares* em aplicativos Android, os que utilizam aprendizado de máquina vêm ganhando destaque [Wu et al., 2021]. Independente de focarem suas abordagens na análise estática, dinâmica ou híbrida, esses trabalhos utilizam as permissões do Android para o desenvolvimento de modelos com bom desempenho de detecção de *malwares* [Alsoghyer and Almomani, 2020]. Entretanto, utilizar todas as 247 permissões das APIs do Android, disponíveis para treinamento dos modelos de aprendizado de máquina, pode representar um desafio de escalabilidade [Yildiz and Doğru, 2019, Li et al., 2018] e impactar negativamente o tempo de execução das soluções.

Com o objetivo de mitigar o problema da escalabilidade, há trabalhos (e.g., [Li et al., 2018, Yildiz and Doğru, 2019]) que investigaram o impacto da redução dos números de permissões utilizadas para o treino dos modelos. Como resultado, verificaram que, mesmo utilizando um número menor de permissões, o tempo de execução pode ser reduzido sem comprometer de forma significativa a acurácia do modelo (i.e., aumento da escalabilidade sem comprometer o desempenho da classificação).

Neste trabalho avaliamos e discutimos a reprodutibilidade e o desempenho do trabalho de [Li et al., 2018, Sun et al., 2016] (SigPID), que pode ser considerado um dos mais relevantes e mais bem citados (mais de 320 citações GSC em setembro de 2021) sobre escalabilidade de modelos de detecção de *malwares* Android. Como o conjunto de dados empregado no trabalho original não está disponível, utilizamos um *dataset* de conhecimento público, que contém 113 permissões. Numa primeira etapa, reproduzimos a redução de características utilizada no SigPID, que consiste em três níveis seleção de permissões. A aplicação dos três níveis de seleção resultou em 27 permissões mais significativas. Em seguida, avaliamos o desempenho do modelo em comparação com diferentes conjuntos de permissões, a saber: (a) 113 permissões (*baseline*) contidas no conjunto de

dados; (b) 22 permissões identificadas no trabalho original do SigPID; (c) 32 permissões mais recorrentes em trabalhos de detecção de *malwares* Android; e (d) 22 permissões classificadas como perigosas pela Google [1]

Como contribuição do trabalho, podemos destacar: (a) a implementação da estratégia de seleção de características empregada pelo SigPID em um conjunto de dados público; (b) a identificação de um subconjunto essencial de permissões (permissões significativas) que pode ser usado para identificar efetivamente *malwares* no Android; (c) um comparativo com o trabalho original, que identifica 22 permissões como significativas e também a comparação com outros conjuntos de dados com diferentes quantidades de permissões; e (d) a análise de aspectos de reprodutibilidade do trabalho original.

O trabalho está organizado como segue. Na Seção 2 apresentamos uma visão geral sobre os requisitos para a reprodução do SigPID e o detalhamos a metodologia de seleção das características. Na Seção 3 discutimos os resultados e apresentamos na Seção 4 as considerações finais.

2. Reprodução dos experimentos

2.1. Detalhamento do Ambiente

Para o desenvolvimento e avaliação dos experimentos, utilizamos um notebook com processador Intel Celeron 1007U (1.5GHz, Dual Core, 2MB L2), 4GB DDR3 1.600MHz, disco rígido de 320GB (SATA - 5.400rpm), Windows 10 Home Single Language, compilação 19042.1110. Para a implementação e avaliação dos modelos, utilizamos as ferramentas Jupyter Notebook (IPython 7.12.0, Python 3.7.6 (default, jan. 8 2020) e o Google Chrome Versão 91.0.4472.124 (Versão oficial) 64 bits. Com exceção do algoritmo *Funtional Tree*, versão 1.0.4, implementado com a ferramenta Weka versão 3.9.5, os demais foram implementados utilizando a versão 0.22.1 da biblioteca Scikit-learn.

Para análise e uso do *dataset*, utilizamos uma divisão estratificada pseudo-aleatória (*test_size*) de 70%/30% [James et al., 2013], a partir dos dados iniciais, sendo 70% utilizado para treinos e 30% para testes. As divisões estratificadas são desejáveis em casos de conjunto de dados desbalanceados, como é o caso do conjunto de dados escolhido.

Para garantir a reprodutibilidade do experimento, definimos arbitrariamente a semente aleatória como 1 para `train_test_split`, de forma a controlar a seleção dos dados de treino e teste. Já os hiperparâmetros, variáveis que controlam o próprio processo de treinamento, foram seguidos conforme o padrão da biblioteca Scikit-learn.

2.2. Dataset

Como mencionado, o conjunto de dados (*dataset*) original do trabalho está indisponível. Para a reprodução e comparação do SigPID com diferentes conjuntos de permissões, selecionamos o conjunto de dados Drebin_215 [Yerima and Sezer, 2018], um sub conjunto do *Drebin project* [Arp et al., 2014], por ser de acesso público e possuir permissões do Android como atributos. O Drebin_215, disponível publicamente no FigShare [2] possui

[1] Ao total, a Google define 30 permissões como perigosas. Destas, 22 estão presentes no conjunto de dados analisado.

[2] `https://figshare.com/articles/dataset/Android_malware\_dataset_for_machine\_learning\_2/5854653`

215 atributos extraídos de 15.036 aplicativos (5.560 malignos e 9.476 benignos), sendo que 113 atributos são permissões.

2.3. Seleção das Permissões através do MLDP

O MLDP (*Multi-Level Data Pruning*) é um método multinível (de 3 níveis) para seleção de características relevantes para uso na construção do modelo. A ideia por trás do método é diminuir o número de permissões (selecionando as mais significativas) e, consequentemente, o tempo de execução dos modelos. O MLDP assume como parâmetro de seleção uma taxa de detecção de *malware* acurácia e precisão de no mínimo 90%, considerada uma taxa ótima.

O método opera nos seguintes três níveis de seleção: (1) classificação de permissão com taxa negativa (*Permission Ranking Negative Rate* ou PRNR); (2) classificação de permissão baseada em suporte (*Support Based Permission Ranking* ou SPR); e (3) mineração de permissões com regras de associação (*Permission Mining with Association Rules* ou PMAR). Cada um dos três níveis é detalhado[3] na versão estendida do trabalho, disponível em [Assolin et al., 2021]. Para reprodutibilidade, os *datasets* e códigos estão disponíveis online no GitHub[4]. Ao final, chegamos a seleção de 108, 30 e 27 permissões nos níveis 1, 2 e 3 de seleção do MLDP, respectivamente.

3. Resultados

Para a reprodução do SigPID, utilizamos o algoritmo SVM, conforme proposto pelos autores no trabalho original [Li et al., 2018]. A Tabela 1 apresenta os resultados da execução do SVM para os diferentes conjuntos de dados, incluindo métricas para a avaliação do desempenho da detecção e o tempo de execução.

Tabela 1. Métricas de avaliação SVM

Conjunto de Dados	Quantidade de Permissões	Precisão	Recall	FPR	F1_Score	Acurácia	Tempo Execução (s)
Nível 1 (PRNR)	108	93,62	88,01	3,52	90,73	93,35	5,44
Nível 2 (SPR)	30	90,03	82,25	5,35	85,96	90,07	2,41
Nível 3 (PMAR)	27	90,13	82,07	5,28	85,91	90,05	2,26
Baseline	113	93,49	87,83	3,59	90,57	93,24	5,84
Perigosas Google	22	86,76	71,88	6,44	78,62	85,55	2,34
Recorrentes	32	88,54	81,53	6,19	84,89	89,27	3,17
SigPID	22	91,77	74,88	3,94	82,47	88,23	2,62

Como podemos observar, os conjuntos de dados que utilizam o mesmo número de permissões obtiveram resultados diferentes. As 22 permissões identificadas no SigPID se destacam em todas as métricas de avaliação quando comparadas com as 22 permissões consideradas perigosas pela Google, ou seja, utilizar permissões perigosas não leva a um melhor resultado qualitativo. Isto indica que a escolha das permissões possui, de fato, um impacto no desempenho na detecção de *malwares*.

[3]O detalhamento inclui uma descrição mais elaborada, as fórmulas matemáticas, as implementações e dados numéricos.

[4]`https://github.com/Malware-Hunter/SigPID/tree/main/wrseg2021`

Quando reduzimos o número de permissões de 113 (*baseline*) para 108 com o nível 1 do MLDP, alcançamos taxas de precisão e acurácia mais altas, acima de 93%. Apesar de haver aumento no *recall*, o F1-Score e a taxa de falsos positivos (FPR) permaneceu mais baixa em relação a *baseline*, assim como o tempo de execução também diminuiu. Quando reduzimos o número de permissões de 113 para 30 com o nível 2 do MLDP, mantiveram-se a acurácia e precisão na faixa de 90%, porém com F1-Score, a métrica a ser considerada, abaixo de 90%. Além disso, obtivemos um ganho de mais de 2 segundos em relação ao tempo de execução.

Além do SVM, avaliamos também outros três algoritmos (*Random Forest*, *Decision Tree* e *Functional Trees* [Gama, 2004]) e comparamos os resultados com os conjuntos de dados discriminados na Tabela 2.

Tabela 2. Conjuntos de dados

Nº de Permissões	Conjunto de dados	Observação
113	*Baseline*	Contidas no *dataset* Drebin_215
22	SigPID	Identificadas no trabalho original do SigPID
32	Recorrentes	Identificadas por meio da interseção de permissões identificadas em outros trabalhos
22	Perigosas	Permissões na lista de perigosas da Google e estavam contidas na *baseline* deste trabalho
27	MLDP	Identificadas após aplicação dos três níveis de seleção sobre a *baseline*

A Tabela 3 sintetiza os resultados dos algoritmos *Random Forest*, *Decision Tree* e *Functional Trees*. As métricas apresentadas são a acurácia, que indica o desempenho geral do modelo, e F1-Score, a média harmônica entre o *recall* e a precisão.

Tabela 3. Métricas de avaliação dos conjuntos de dados

	Decision tree		Random forest		Functional Trees	
Conjunto de dados	F1_Score	Acurácia	F1_Score	Acurácia	F1_Score	Acurácia
113 Baseline	92,28	94,32	94,44	95,94	97,30	97,27
22 SigPID	88,46	92,00	89,30	92,57	92,90	93,05
27 MLDP	91,09	93,50	92,62	94,64	95,40	95,43
32 Recorrentes	90,89	93,44	92,46	94,57	95,60	95,63
22 Perigosas Google	81,62	87,74	85,31	88,85	89,10	89,02

Quatro dos cinco conjuntos de dados obtiveram acurácia acima dos 90% (*Baseline*, SigPID, MLDP e Recorrentes). Destes, o *Baseline* performou melhor que os demais. Quando utilizamos o *Decision Tree* atingimos 94,32% de acurácia e 92,28% de F1-Score. Com os algoritmos *Random Forest* e *Functional Trees* atingimos 95,94% de acurácia e 94,44% de F1-Score e acurácia e F1-Score acima dos 97%, respectivamente.

Utilizando 22 permissões do SigPID a acurácia se manteve na faixa de 92% com *Decision Tree* e *Random Forest*, já F1-Score se manteve abaixo de 90%. O *Functional Trees* foi o algoritmo com melhor desempenho, atingindo 93,05% acurácia e 92,90% de F1-Score. Segundo os dados da Tabela 3, os conjuntos de dados MLDP e Recorrentes obtiveram resultados muito próximos, ficando ambos com acurácia acima de 95% utilizando o *Functional Trees*, por exemplo.

Na Figura 1 apresentamos os dados sobre o tempo de execução de cada algoritmo. Como podemos observar, o *Baseline* tem o maior tempo de execução, chegando a 11,31 segundos com algoritmo *Functional Trees*. Entretanto, é interessante observar que o problema ocorre apenas para o *Baseline*, já que o algoritmo executa em menos tempo que alguns dos demais para conjuntos de dados menores.

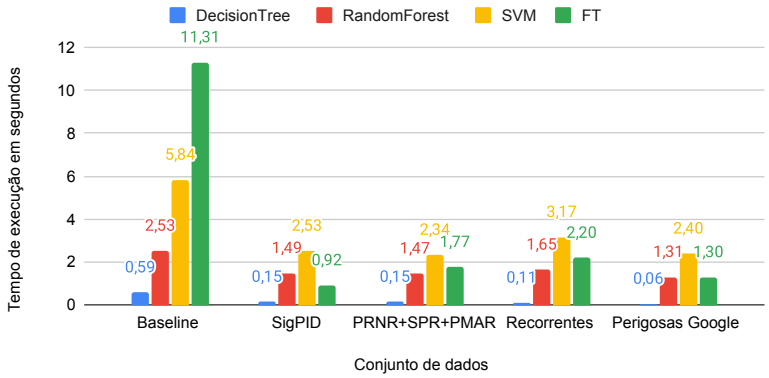

Figura 1. Tempo de execução para diferentes conjuntos de dados

O *Functional Trees* é um algoritmo baseado em árvores de classificação que podem ter funções de regressão logística nos nós internos ou folhas. O algoritmo pode lidar com variáveis de destino binárias e multi-classe, atributos numéricos, nominais e valores ausentes, o que pode aumentar o tempo de execução de acordo com o número de características de entrada [Gama, 2004]. Como a estrutura do algoritmo é uma generalização das *Multivariate Trees* [Gama, 2001], sua complexidade pode ser similar, isto é, $\mathcal{O}(n^2)$. Esta complexidade pode ser utilizada para explicar o comportamento apresentado no gráfico da Figura 1. De fato, o algoritmo *Functional Trees* reduziu significativamente o tempo de execução para conjuntos de dados menores (e.g., 1,77s para as 27 permissões do MLDP).

Ao analisarmos os conjuntos de dados que utilizam 27 e 32 permissões, observa-se que o tempo de execução e as métricas estão muito próximas para o *Functional Trees*. Nesse caso, podemos dizer que os modelos são equivalentes, isto é, não faz diferença utilizar as 32 permissões recorrentes ou as 27 permissões do MLDP. Isto prova que a identificação das permissões recorrentes, em trabalhos existentes na literatura, leva a resultados tão bom quanto os resultados do método de múltiplos níveis de seleção do SigPID, o que confirma uma das nossas hipóteses, isto é, permissões recorrentes podem ter um impacto positivo sobre os modelos de detecção de *malwares*.

4. Considerações Finais

Aplicando os 3 níveis de seleção do SigPID nas 113 permissões do *dataset* Drebin_215, conseguimos reduzir em 76% o número de permissões a serem analisadas para detecção de *malwares* Android, mantendo acurácia acima de 90% com SVM, 93,50% com *Decision Tree*, 94,64% com *Random Forest* e 95% com o *Functional Trees*. Além disso, conseguimos também reduzir o tempo de execução dos modelos, porém, ao custo de um leve aumento na taxa de falsos positivos.

Além das 27 permissões resultantes dos 3 níveis de seleção, utilizamos também conjuntos de 113, 22 e 32 permissões. Percebemos que o conjunto de dados que utiliza todas permissões (113) foi o que melhor performou (e.g., 97% de acurácia), porém, ao preço de um tempo de execução significativamente maior. Um caso interessante foi quando comparamos os conjuntos com 22 permissões, sendo um oriundo do trabalho original do SigPID e 22 permissões classificadas como perigosas pela Google. O SigPID chegou a 93% de acurácia enquanto que as perigosas mantiveram acurácia abaixo de 90%, o que indica que o fato da permissão ser classificada como perigosa não a torna necessariamente relevante para detecção de *malwares*. Outra observação interessante é o fato de os conjuntos de dados com 32 permissões mais recorrentes e as 27 identificadas atingirem resultados muito próximos ao aplicar os 3 níveis de seleção. Isto indica que escolher as permissões de acordo com a recorrência pode ser um caminho de investigação a ser seguido.

Como trabalhos futuros, podemos elencar: (a) testes com conjuntos de dados maiores; (b) testes com conjuntos de dados atuais; (c) avaliação dos níveis de seleção para outras características (e.g., *intents* e chamadas de API); (d) otimização de hiperparâmetros; (e) testar os modelos em *smartphones* modernos; e (f) mensurar o tempo de execução dos modelos em CPUs modernas projetadas para acelerar a computação de algoritmos de aprendizado de máquina.

Referências

Alsoghyer, S. and Almomani, I. (2020). On the effectiveness of application permissions for android ransomware detection. In *2020 6th Conference on Data Science and Machine Learning Applications (CDMA)*, pages 94–99.

Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K., and Siemens, C. (2014). Drebin: Effective and explainable detection of android malware in your pocket. In *Ndss*, volume 14, pages 23–26.

Assolin, J., Rocha, V., Kreutz, D., Siqueira, G., Rodrigues, G., Feitosa, E., and Casola, K. (2021). Detecção de Malwares Android: reprodução da seleção de características do SigPID. `https://arxiv.kreutz.xyz/wrseg21_sigpid_ve1.pdf`.

Gama, J. (2001). Functional trees for classification. In *Proceedings 2001 IEEE International Conference on Data Mining*, pages 147–154. IEEE.

Gama, J. (2004). Functional trees. *Machine learning*, 55(3):219–250.

James, G., Witten, D., Hastie, T., and Tibshirani, R. (2013). *An introduction to statistical learning*, volume 112. Springer.

Li, J., Sun, L., Yan, Q., Li, Z., Srisa-an, W., and Ye, H. (2018). Significant permission identification for machine-learning-based android malware detection. *IEEE Transactions on Industrial Informatics*, 14(7):3216–3225.

Sun, L., Li, Z., Yan, Q., Srisa-an, W., and Pan, Y. (2016). Sigpid: significant permission identification for android malware detection. In *2016 11th international conference on malicious and unwanted software (MALWARE)*, pages 1–8. IEEE.

Wu, Q., Zhu, X., and Liu, B. (2021). A survey of android malware static detection technology based on machine learning. *Mobile Information Systems*, 2021.

Yerima, S. Y. and Sezer, S. (2018). Droidfusion: A novel multilevel classifier fusion approach for android malware detection. `https://figshare.com/articles/dataset/Android_malware_dataset_for_machine_learning_2/5854653`.

Yildiz, O. and Doğru, I. A. (2019). Permission-based android malware detection system using feature selection with genetic algorithm. *International Journal of Software Engineering and Knowledge Engineering*, 29(02):245–262.

APÊNDICE C – PERMISSÕES RESULTANTES DE CADA NÍVEL DE CORTE DO MLDP

PRNR	
SEND_SMS	BIND_REMOTEVIEWS
DELETE_CACHE_FILES	READ_CALL_LOG
WRITE_APN_SETTINGS	READ_SOCIAL_STREAM
INSTALL_PACKAGES	WRITE_PROFILE
ACCESS_LOCATION_EXTRA_COMMANDS	ADD_VOICEMAIL
WRITE_HISTORY_BOOKMARKS	WRITE_USER_DICTIONARY
RECEIVE_SMS	WRITE_CALL_LOG
UPDATE_DEVICE_STATS	BIND_TEXT_SERVICE
READ_SMS	BIND_VPN_SERVICE
WRITE_SMS	READ_PROFILE
READ_HISTORY_BOOKMARKS	WRITE_SOCIAL_STREAM
CONTROL_LOCATION_UPDATES	AUTHENTICATE_ACCOUNTS
DELETE_PACKAGES	NFC
RECEIVE_WAP_PUSH	READ_SYNC_STATS
RESTART_PACKAGES	USE_CREDENTIALS
PROCESS_OUTGOING_CALLS	SUBSCRIBED_FEEDS_WRITE
BIND_WALLPAPER	MANAGE_ACCOUNTS
CLEAR_APP_USER_DATA	READ_USER_DICTIONARY
RECEIVE_MMS	CHANGE_WIFI_MULTICAST_STATE
READ_PHONE_STATE	MASTER_CLEAR
SET_WALLPAPER	SET_TIME
HARDWARE_TEST	WRITE_GSERVICES
CHANGE_CONFIGURATION	DUMP
MOUNT_UNMOUNT_FILESYSTEMS	SET_TIME_ZONE
RECEIVE_BOOT_COMPLETED	BIND_ACCESSIBILITY_SERVICE
READ_LOGS	READ_SYNC_SETTINGS
SET_PREFERRED_APPLICATIONS	SET_PROCESS_LIMIT
CALL_PHONE	INSTALL_LOCATION_PROVIDER
ACCESS_COARSE_LOCATION	WRITE_SYNC_SETTINGS
MODIFY_PHONE_STATE	SUBSCRIBED_FEEDS_READ
DISABLE_KEYGUARD	BIND_INPUT_METHOD
CHANGE_WIFI_STATE	CALL_PRIVILEGED
ACCESS_FINE_LOCATION	BROADCAST_STICKY
CLEAR_APP_CACHE	REORDER_TASKS
ACCESS_WIFI_STATE	SET_ACTIVITY_WATCHER
WRITE_EXTERNAL_STORAGE	MODIFY_AUDIO_SETTINGS
READ_CONTACTS	MOUNT_FORMAT_FILESYSTEMS
GET_PACKAGE_SIZE	RECORD_AUDIO
WRITE_SECURE_SETTINGS	BIND_APPWIDGET
DEVICE_POWER	GET_ACCOUNTS
ACCESS_NETWORK_STATE	READ_CALENDAR
WRITE_CONTACTS	CAMERA
CHANGE_COMPONENT_ENABLED_STATE	ACCESS_SURFACE_FLINGER
SET_WALLPAPER_HINTS	PERSISTENT_ACTIVITY
SYSTEM_ALERT_WINDOW	BROADCAST_WAP_PUSH

VIBRATE	REBOOT
WRITE_SETTINGS	WRITE_CALENDAR
CHANGE_NETWORK_STATE	GLOBAL_SEARCH
WAKE_LOCK	BIND_DEVICE_ADMIN
ACCESS_MOCK_LOCATION	BATTERY_STATS
EXPAND_STATUS_BAR	INTERNAL_SYSTEM_WINDOW
GET_TASKS	BLUETOOTH
SET_ORIENTATION	READ_EXTERNAL_STORAGE
FLASHLIGHT	READ_FRAME_BUFFER

PRNR+SPR	
ACCESS_NETWORK_STATE	CHANGE_WIFI_STATE
WRITE_EXTERNAL_STORAGE	WRITE_SETTINGS
READ_PHONE_STATE	CAMERA
WAKE_LOCK	CALL_PHONE
ACCESS_WIFI_STATE	WRITE_SMS
RECEIVE_BOOT_COMPLETED	WRITE_CONTACTS
VIBRATE	READ_EXTERNAL_STORAGE
GET_ACCOUNTS	MANAGE_ACCOUNTS
ACCESS_FINE_LOCATION	USE_CREDENTIALS
ACCESS_COARSE_LOCATION	READ_HISTORY_BOOKMARKS
SEND_SMS	CHANGE_NETWORK_STATE
READ_CONTACTS	RECORD_AUDIO
RECEIVE_SMS	READ_SYNC_SETTINGS
READ_SMS	RESTART_PACKAGES
GET_TASKS	BLUETOOTH

PRNR+SPR+PMAR	
ACCESS_NETWORK_STATE	CHANGE_WIFI_STATE
WRITE_EXTERNAL_STORAGE	WRITE_SETTINGS
READ_PHONE_STATE	CAMERA
WAKE_LOCK	CALL_PHONE
RECEIVE_BOOT_COMPLETED	WRITE_CONTACTS
VIBRATE	READ_EXTERNAL_STORAGE
GET_ACCOUNTS	USE_CREDENTIALS
ACCESS_FINE_LOCATION	READ_HISTORY_BOOKMARKS
ACCESS_COARSE_LOCATION	CHANGE_NETWORK_STATE
SEND_SMS	RECORD_AUDIO
READ_CONTACTS	READ_SYNC_SETTINGS
RECEIVE_SMS	RESTART_PACKAGES
READ_SMS	BLUETOOTH
GET_TASKS	