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 propõe 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, utilizá-las ainda é um desafio uma vez que existe um grande número de permissões disponíveis no sistema Android, e isto 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 pode não ser comprometida e o tempo de execução é reduzido (*i.e.*, 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 e 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 INTERNET, RECEIVE_BOOT_COMPLETED, ACCESS_NETWORK_STATE, ACCESS_WIFI_STAT, KILL_BACKGROUND_PROCESSES, CHANGE_NETWORK_STATE e WAKE_LOCK, apesar de não serem consideradas perigosas, aparecem com frequência em *malwares*. Isto acontece 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]

[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.

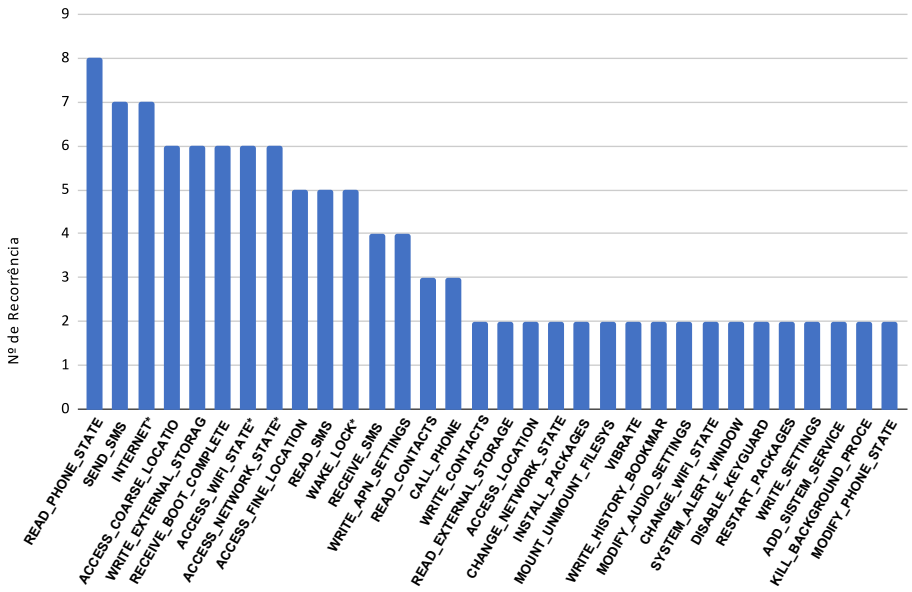

Figura 1. Permissões recorrentes

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, como pode ser observado em um estudo recente realizado pelos autores [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 utili-

zados. Sem os hiper-parâmetros utilizados nos modelos, que são variáveis de configuração que controlam o processo de treinamento, é tecnicamente muito difícil de reproduzir os resultados. Em síntese, a presença desses problemas são fatores que impossibilitam a reprodução fidedigna do estudo.

4. Estratégia Adotada

A primeira estratégica 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 são variáveis que controlam o próprio processo de treinamento e 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.

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

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**	**precision**	**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% em relação a 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, mas, ao mesmo tempo, 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 antiga (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.