Uma Avaliação Hibrida para Identificação de Ameaças no Android

Rodrigo Lemos, Tiago Heinrich

Departamento de Informática
Universidade Federal do Paraná (UFPR)
Curitiba – PR – Brasil

1. Introdução

O Android é um sistema operacional *open source* que tem a maior presença no mercado de *smartphones* [Mohini et al. 2013]. Fundado em 2003, o sistema é baseado no kernel lts do Linux, sendo mantido pela *Open Handset Alliance (OHA)* a qual é responsável por definir padrões de desenvolvimento para dispositivos móveis.

Ao considerar a segurança presente no sistema, algumas medidas tem destaque como, prevenir que aplicações ganhem acesso crítico do sistema operacional, e permissões devem ser requisitadas para o usuário antes de uma aplicação ter acesso a certo tipo de recurso [Yeon et al. 2016]. Mas apesar das medidas de segurança já tomadas problemáticas como avaliações limitadas de aplicativos que estão na `loja de aplicativos`, aplicativos tem a liberdade de acesso de informações entre si, livre acesso de informações no dispositivo, dentre outros.

A partir destas problemas modelos de segurança veem sendo desenvolvidos para verificar e buscar ameaças que possam vir a prejudicar o sistema. Apesar da alta capacidade de processamento que *smartphones* atuais possuem, ainda existe dependências de certos recursos como baterias.

Assim uma estratégia para identificação de ameaças é o uso de *Intrusion Detection System (IDS)*, que pode realizar atividades como coleta de dados, tratamento de dados, monitoramento e tomada de decisões [Bridges et al. 2020]. Para o monitoramento e tomada de decisão algoritmos ou heurísticas são exploradas para a identificação de ameaças através de padrões ou características conhecidas. Estas estratégias podem variar desda utilização de aprendizado de máquina, processos estatísticos, comparação com assinaturas conhecidas, dentre outros.

O respectivo estudo visa solucionar limitações em modelos anteriores, através da utilização de um modelo hibrido para a identificação de ameaças no ambiente Android. A nova estratégia proposta consiste em explorar a comunicação entre processos realizadas pelo *binder* para a identificação de ameaças [Yeon et al. 2016], ao qual ainda não foi explorado na literatura.

A base de dados utilizadas para o treinamento dos modelos de aprendizado de máquina, consiste de base de dados estática Androzoo e dinâmica AndroDi. Um subconjunto de *features* de aplicações do Androzoo [Allix et al. 2016] foi explorado para completar os dados dinâmicos (ainda não publicada, Autor Fabricio Ceschin). Já o AndroDi é uma base de dados (ainda não publicada, Autor Rodrigo Lemos), que consiste de observações de processos no Android através do *binder*.

Este trabalho está estruturado em cinco seções. A Seção 2 descreve os principais conceitos para o estudo. Na Seção 3 são apresentados os trabalhos que têm sido realizados neste contexto. A técnica proposta e estratégia de avaliação é apresentada na Seção 4. A Seção 5 apresenta a discussão dos experimentos. Por fim, a Seção 6 traz as conclusões do trabalho.

2. Fundamentação Teórica

2.1. Binder

O *binder* é responsável por realizar toda a comunicação entre processos do Android, aos quais devem passar por ele. Por consequência modelos de análise estática acabam utilizando o *binder*, extraindo informações como Intents e permissões como características. Este sendo um ponto crítico para ataques, conforme apresentando na literatura [Artenstein and Revivo 2014, Shen et al. 2017].

2.2. AndroDi

O AndroDi visa definir comportamento de aplicativos, estes levam em consideração os aplicativos presentes no Androzoo [Allix et al. 2016]. A base de dados consiste em um conjunto de traços das chamadas de função no *binder*, onde cada traço representa o comportamento de um aplicativo diferente analisado.

Com a base de dados Androzoo, são selecionados aplicativos maliciosos e normais para serem executados em um emulador Android enquanto são monitorados para capturar o traço das chamadas de função do *binder*. Para estimular o funcionamento dos aplicativos analisados enquanto são monitorados, são injetados 1000 eventos nos mesmos por meio da ferramenta Monkey [Google 2019] como forma de simular a interação do usuário. Após o traço ser gerado, ele é extraído para o computador, o aplicativo analisado é desinstalado e o processo é repetido para o próximo aplicativo.

O traço são convertidos para produzir uma representação esparsa das contagens, as quais são utilizadas para representar as operações capturadas para cada aplicação observada no *binder*. Atualmente o *dataset* conta com 201 apks distintos.

2.3. Subset Androzoo

O subset androzoo é um dataset estático, que contem 25 atributos aos quais foram extraídos utilizando a API do Androzoo. Deste conjunto total somente oito são atributos textuais, aos quais não foram utilizados neste respectivo estudo.

Devido a distribuição de Apks encontradas no AndroDi não foi possível aproveitar as informações já presentes, que foram coletadas usando a API, já que o conjunto de apks era mais recente em relação ao subset do androzoo. Por este motivo, foi implementado um *script* que é responsável por receber um apk e fazer a extração de todas as *features* presentes no subset androzoo. Este permite a extração das *features* as quais posteriormente podem ser utilizadas para adicionar novas amostra no subset androzoo.

3. Trabalhos Relacionados

SAMADroid se trata de um detector de análise híbrida proposto por [Arshad et al. 2018] com a intenção de apresentar um bom desempenho na precisão e nos consumos de bateria

e armazenamento. Para tal, o modelo proposto se estrutura em três níveis: coleta de atributos estáticos e dinâmicos, comunicação entre *hosts* local e remoto e algoritmo de aprendizado de máquina. Como atributos dinâmicos são utilizados os logs de chamadas de sistema gerados no *host* local e enviados para o servidor remoto e como atributos estáticos são usados dados do AndroidManifest.xml e do código SMALI coletados no servidor remoto onde o aplicativo é decompilado. O servidor remoto segue analisando em tempo real os logs de chamadas de sistema enviados pelo *host* local juntamente com os atributos estáticos através de algoritmos de aprendizado de máquina. Essa abordagem de análise remota permite ao SAMADroid analisar em tempo real com um *overhead* mínimo nos recursos de bateria e memória, embora tenha como limitação a necessidade de comunicação constante entre o *host* local e o servidor remoto para funcionar. Como resultado, o modelo alcançou uma alta precisão na detecção com um baixo índice de falsos positivos.

[Yerima and Sezer 2019] propôs uma arquitetura multinível para combinar diferentes algoritmos de classificação que utilizam aprendizado de máquina, chamada DroidFusion. O framework foi proposto para combinar os resultados de classificadores genéricos e logo é independente dos atributos utilizados pelos mesmos, embora tenha sido testado apenas para algoritmos estáticos durante esse estudo. A fusão dos algoritmos é implementada através de um algoritmo que se baseia na precisão de cada algoritmo do nível abaixo para ranqueá-los e então gerar a classificação definitiva com base em suas saídas. Em comparação com os frameworks que utilizam uma arquitetura em pilha para combinar algoritmos, o DroidFusion se mostrou mais preciso em todos os testes realizados, além disso o framework precisou em média de sete segundos para analisar um aplicativo e portanto possui um desempenho razoável para ser utilizado em produção.

Embora [Alzaylaee et al. 2020] tenha proposto uma abordagem para análise dinâmica, os melhores resultados de seu modelo foram encontrados ao combinar atributos dinâmicos e as permissões do aplicativo, sendo assim a contribuição de sua pesquisa é um modelo de análise híbrida. O modelo proposto consiste em coletar dados dinamicamente ao simular o uso do aplicativo em uma sandbox de modo *stateful*, ou seja o simulador leva em conta o estado atual do aplicativo para executar a próxima ação, e então analisar esses dados, juntamente com os dados de permissões, através de *Deep Learning*. Como resultados, [Alzaylaee et al. 2020] concluiu que a coleta *stateful* de atributos dinâmicos é mais eficiente que a *stateless* e que o uso de atributos híbridos é mais eficiente que utilizar apenas atributos dinâmicos através da comparação entre essas abordagens. Além disso, o modelo obteve alta taxa de detecções com um número de falsos positivos baixo, se mostrando assim uma abordagem eficiente.

4. Proposta

No ambiente Android uma estratégia utilizada por atacantes consiste na ofuscação de código, que têm um grande impacto na eficiência de mecanismos para a detecção de ameaças, diminuindo as suas taxas de detecção em mais de 50% na média para mecanismos de análise estática [Ajiri et al. 2020].

Desta forma, o respectivo estudo visa explorar informações dinâmicas do *binder*, além de dados estáticos, para a identificação de ameças em aplicações do Android. Visando otimizar a identificação de ameaças no ambiente as quais não sejam afetadas por

técnicas de ofuscação e obtendo um novo ponto de visão dentro do sistema, ao qual ainda não foi retratado na literatura.

O modelo hibrido que consiste na combinação de features provenientes de traços de execução do *binder* e features estáticas provenientes dos *apks*, principalmente do código fonte e do *AndroidManifest.xml*. Estas *features* são características textuais extraídas diretamente dos *apks* utilizados para o respectivo experimento, como por exemplo pacotes do apk, permissões, etc. Já o dataset dinâmico consiste no histograma de comandos binder chamados durante a execução dos *apks*.

O *dataset* hibrido contém sete *features* estáticas que foram extraídas de acordo com a definição do [Fabricio] e trinta e sete *features* dinâmicas que visam representar a execução de cada respectivo *apk*.

Por se tratarem de informações textuais, mais especificamente de listas, algumas features do Androzoo foram representadas em várias features que representavam a presença de cada um dos elementos dessas listas no respectivo *apk*. Dessa forma, o *dataset* resultante desse processo tinha 4344 *features*.

Estas características foram avaliadas antes da execução do modelo e uma seleção manual foi efetuada para remover *features* as quais não traziam nenhuma contribuição para o modelo, possuíam informação faltando, representavam informações as quais não eram relevantes para o modelo e informações em um formato inválido. Após esta seleção, foi efetuado uma seleção de *features* utilizando o `SelectPercentile` que realiza o calculo do percentil e seleciona os maiores p-values de acordo com o método ANOVA.

Este método de seleção apresentou resultados adequados em decorrência da base de dados dinâmica que acaba possuindo uma distribuição maior de valores em relação ao conteúdo encontrado nos dados estáticos. A normalização utiliza o método `MinMaxScaler` para transformar as *features* nos intervalos adequados.

Para a avaliação foi utilizado os algoritmos *naive Bayes*, *KNeighbors* com um k igual a 5, *random forest* com um número de árvores de 100, *ada Boost* com um estimador em 100, *linear SVC* e *multilayer perceptron* com um número máximo de iterações de 300.

5. Avaliação Experimental

O dataset foi divido, conforme orientado, em 80% para treino e 20% para teste. Após a divisão do conjunto, os dados de treino foram utilizados para treinar os modelos com uma divisão de 80% para treino e 20% para validação, com resultados apresentados nessa seção, e utilizando a validação cruzada k-fold. Os dados de teste foram classificados por todos os modelos treinados para avaliar os resultados. Na Seção 5.1 e 5.2 são apresentados os resultados avaliação cruzada com o kFold na base de validação e na base de testes respectivamente. Os gráficos de curva ROC de cada um dos resultados apresentados encontra-se na 7

Após este processo foi realizado a seleção de *features* onde foram escolhidas 434 *features* das 4,344. A Figura 1 e 2 apresenta a distribuição das *features* para cada partição do *dataset*, apesar do número pequeno de amostras ainda é possível destacar uma semelhança entre as duas partições. Sendo que classes com alta presença na Figura 1 ainda tem predominância na Figura 2.

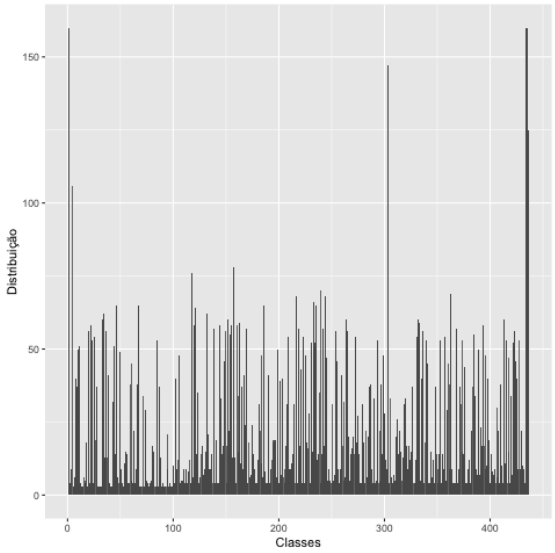

Figura 1. Distribuição de amostras por classe 80%.

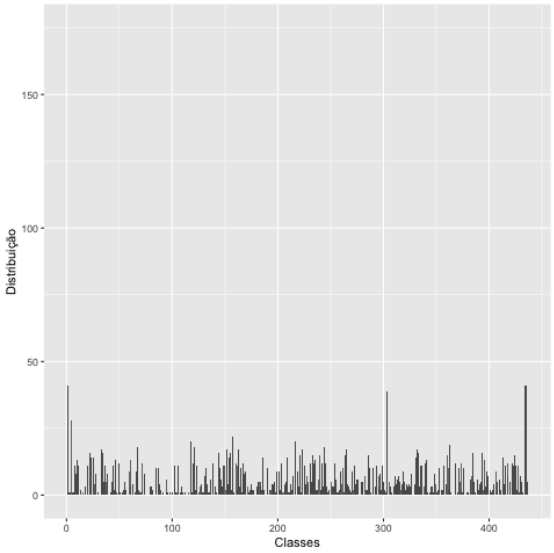

Figura 2. Distribuição de amostras por classe 20%.

A Tabela 1 apresenta o primeiro conjunto de testes, composto por seis algoritmos (as quais somente um é one-class, todos os outros sendo multi-class). Na primeira coluna da Tabela 1 é apresentado a matriz de confusão para cada algoritmo e na segunda coluna são apresentadas seis métricas de avaliações escolhidas para determinar a qualidade dos experimentos e *dataset*.

As matrizes de confusão destaca uma baixa taxa de erro, ao qual o algoritmo poderia errar a classe que deseja classificar. O único algoritmo que apresenta um fator alto de Falsos Positivos é o *Linear SVC* que também reflete os piores resultados em relação ao F1Score e curva ROC. Todos outros modelos apresentam uma taxa ROC e F1Score superior a 0.83, demonstrando um aprendizado adequado considerando os dados de treinamento. Também é possível destacar uma taxa Bier e *Mean Absolute Error* inferior a 0.15, que reforça os resultados da curva ROC (os respectivos gráficos podem ser encontrados no apêndice do trabalho).

5.1. kFold 80%

A Tabela 2 apresenta os resultados para validação cruzada utilizando o k-fold com 80% dos dados amostrais. A primeira partição ($Alg.$) apresenta o algoritmo, a segunda partição apresenta o *k-fold* para cada um dos cinco *fold*, as ultimas duas partições da tabela apresentam a matriz de confusão (TP TruePositive, FP FalsePositive, FN FalseNegative, TN TrueNegative) as métricas de avaliação respectivamente (seguindo a definição da Tabela 1).

O comportamento observado na Tabela 2 é semelhante aos resultados encontrados na Tabela 1. A distribuição de falsos positivos é pequena, com um F1Score e curva ROC adequado para cinco dos seis algoritmos explorados.

Matriz de confusão		Métricas de avaliação					
Naive Bayes							
TP 14	FP 3	F1Score	0.84	Recall	0.875	Precision	0.82
FN 2	TN 13	MAE	0.15	Brier	0.15	ROC AUC	0.84
KNeighbors							
TP 15	FP 3	F1Score	0.88	Recall	0.94	Precision	0.83
FN 1	TN 13	MAE	0.12	Brier	0.12	ROC AUC	0.87
Random Forest							
TP 13	FP 0	F1Score	0.90	Recall	0.81	Precision	1.00
FN 3	TN 16	MAE	0.09	Brier	0.09	ROC AUC	0.91
Ada Boost							
TP 13	FP 0	F1Score	0.90	Recall	0.81	Precision	1.00
FN 3	TN 16	MAE	0.09	Brier	0.09	ROC AUC	0.91
Linear SVC							
TP 14	FP 3	F1Score	0.85	Recall	0.87	Precision	0.82
FN 2	TN 13	MAE	0.16	Brier	0.16	ROC AUC	0.84
Multilayer Perceptron							
TP 14	FP 0	F1Score	0.93	Recall	0.87	Precision	1.00
FN 2	TN 16	MAE	0.06				

Tabela 1. Primeira execução sem o kFold. MAE é a abreviação para `Mean Absolute Error`, e Brier representa `Brier Score`.

5.2. kFold 20%

A Tabela 3 apresenta os resultados com a base de testes utilizando o *k-fold*. Os testes sem utilização de validação cruzada foram ocultados por sua semelhança com os folds analisados. A distribuição dos resultados acaba sendo diferente em relação aos casos apresentados na Tabela 1 e 2, onde somente dois modelos acabam mantendo um resultado positivo estes sendo *Random Forest* e *Ada Boost*. Os resultados de *Naive Bayes* e *Multilayer Perceptron* apresentam uma taxa de falso positivos e falsos ngavitos alto em comparação aos resultados anteriores, e o resultado positivo encontrado na curva ROC acaba sendo prejudicado por esta taxa de erro. Por fim os algoritmos com maior impacto pelo erro na classificação foram o *Linear SVC* e *KNeighbros*.

6. Conclusão

Este trabalho apresentou uma estratégia hibrida para a identificação de ameaças no Android. O estudo acaba utilizando dois *datasets* para a extração de informações aos quais são utilizadas para treinar os modelos.

Os resultados destacam a possibilidade da utilização de técnicas hibridas para a identificação de ameaças, mas ainda é possível destacar que a avaliação usando o *k-fold* poderia ter melhores resultados caso um conjunto maior de dados fosse utilizado. O *dataset* atual consta somente 201 apks, que ao serem repartidos em inúmeros blocos com tamanho relativamente pequeno acabam afetando um dos conjuntos de teste, prejudicando resultados positivos encontrados nos outros dois casos de teste.

Por fim, é possível destacar que apesar de alguns modelos não terem resultados agráveis na base de teste, foi possível manter um resultado adequando ao utilizar o *Random Forest* e *Ada Boost*, que mantém uma boa distribuição na sua matriz de confusão.

Alg.	kfold	TP	FP	FN	TN	F1Score	Recall	Precision	MAE	Brier	ROC
Naive Bayes	1	14	3	2	13	0.85	0.87	0.82	0.16	0.16	0.84
	2	13	3	1	15	0.87	0.93	0.81	0.12	0.12	0.88
	3	17	2	0	13	0.94	1.0	0.89	0.06	0.06	0.93
	4	12	1	1	18	0.92	0.92	0.92	0.06	0.06	0.93
	5	19	1	4	8	0.88	0.83	0.95	0.15	0.15	0.85
KNeighbors	1	15	3	1	13	0.88	0.93	0.83	0.12	0.12	0.87
	2	14	5	0	13	0.85	1.0	0.74	0.16	0.16	0.86
	3	17	1	0	14	0.97	1.0	0.94	0.3	0.3	0.97
	4	13	1	0	18	0.96	1.00	0.93	0.03	0.03	0.97
	5	19	2	4	7	0.86	0.82	0.90	0.18	0.18	0.80
Random Forest	1	13	0	3	16	0.90	0.81	1.00	0.09	0.09	0.91
	2	13	1	1	17	0.92	0.92	0.92	0.06	0.06	0.93
	3	16	1	1	14	0.94	0.94	0.94	0.06	0.06	0.93
	4	13	2	0	17	0.93	1.00	0.87	0.06	0.06	0.95
	5	19	0	4	9	0.90	0.83	1.00	0.12	0.12	0.91
Ada Boost	1	13	0	3	16	0.90	0.81	1.00	0.09	0.09	0.91
	2	13	1	1	17	0.92	0.92	0.92	0.06	0.06	0.93
	3	16	1	1	14	0.94	0.94	0.94	0.06	0.06	0.93
	4	13	2	0	17	0.93	1.00	0.87	0.06	0.06	0.95
	5	19	0	4	9	0.90	0.83	1.00	0.12	0.12	0.91
Linear SVC	1	14	3	2	13	0.85	0.87	0.82	0.16	0.16	0.84
	2	14	2	0	16	0.93	1.0	0.87	0.06	0.06	0.94
	3	16	2	1	13	0.91	0.94	0.89	0.09	0.09	0.90
	4	13	3	0	16	0.90	1.0	0.81	0.09	0.09	0.92
	5	19	3	4	6	0.84	0.83	0.86	0.22	0.22	0.75
Multilayer Perceptron	1	14	0	2	16	0.93	0.87	1.00	0.06		
	2	13	3	1	15	0.87	0.93	0.81	0.12		
	3	16	1	0	15	0.97	0.94	1.0	0.03		
	4	13	2	0	17	0.93	1.00	0.87	0.06		
	5	19	1	4	8	0.88	0.83	0.95	0.15		

Tabela 2. Execução com o k-fold utilizando 80% da base de dados. MAE é a abreviação para `Mean Absolute Error`, e Brier representa `Brier Score`.

Alg.	kfold	TP	FP	FN	TN	F1Score	Recall	Precision	MAE	Brier	ROC
Naive Bayes	1	5	0	17	19	0.37	0.22	1.00	0.41	0.41	0.61
	2	5	0	17	19	0.37	0.22	1.00	0.41	0.41	0.61
	3	5	0	17	19	0.37	0.22	1.00	0.41	0.41	0.61
	4	6	0	16	19	0.43	0.27	1.0	0.39	0.39	0.64
	5	4	0	18	19	0.31	0.18	1.0	0.44	0.44	0.59
KNeighbors	1	0	1	22	18	0.0	0.0	0.0	0.56	0.56	0.47
	2	0	0	22	19	0.0	0.0	0.0	0.54	0.54	0.50
	3	0	0	22	19	0.0	0.0	0.0	0.54	0.54	0.50
	4	0	0	22	19	0.0	0.0	0.0	0.54	0.54	0.50
	5	0	0	22	19	0.0	0.0	0.0	0.54	0.54	0.50
Random Forest	1	18	3	4	16	0.83	0.81	0.85	0.17	0.7	0.83
	2	18	3	4	16	0.83	0.81	0.85	0.17	0.7	0.83
	3	18	2	4	17	0.85	0.81	0.9	0.14	0.14	0.85
	4	18	3	4	16	0.83	0.81	0.85	0.17	0.7	0.83
	5	18	2	4	17	0.85	0.81	0.9	0.14	0.14	0.85
Ada Boost	1	18	3	4	16	0.83	0.81	0.85	0.17	0.17	0.83
	2	18	3	4	16	0.83	0.81	0.85	0.17	0.17	0.83
	3	18	2	4	17	085	0.81	0.9	0.14	0.14	0.85
	4	18	2	4	17	0.85	0.81	0.9	0.14	0.14	0.85
	5	18	2	4	17	0.85	0.81	0.9	0.14	0.14	0.85
Linear SVC	1	0	0	22	19	0.0	0.0	0.0	0.53	0.53	0.5
	2	0	0	22	19	0.0	0.0	0.0	0.53	0.53	0.5
	3	0	0	22	19	0.0	0.0	0.0	0.53	0.53	0.5
	4	0	0	22	19	0.0	0.0	0.0	0.54	0.54	0.50
	5	0	0	22	19	0.0	0.0	0.0	0.53	0.53	0.5
Multilayer Perceptron	1	22	19	0	0	0.69	1.0	0.53	0.46	0.46	0.5
	2	8	8	14	11	0.42	0.33	0.5	0.53	0.53	0.47
	3	6	7	16	12	0.34	0.27	0.46	0.56	0.56	0.45
	4	0	0	22	19	0.0	0.0	0.0	0.54	0.54	0.50
	5	0	0	22	19	0.0	0.0	0.0	0.54	0.54	0.50

Tabela 3. Execução com o k-fold utilizando 20% da base de dados. MAE é a abreviação para `Mean Absolute Error`, e Brier representa `Brier Score`.

Agradecimentos

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Código de Financiamento 001. Os autores também agradecem aos departamentos de Informática da UFPR.

Referências

Ajiri, V., Butakov, S., and Zavarsky, P. (2020). Detection efficiency of static analyzers against obfuscated android malware. In *2020 IEEE 6th Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing, (HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS)*. IEEE.

Allix, K., Bissyandé, T. F., Klein, J., and Le Traon, Y. (2016). Androzoo: Collecting millions of android apps for the research community. In *Proceedings of the 13th International Conference on Mining Software Repositories*, MSR '16, pages 468–471, New York, NY, USA. ACM.

Alzaylaee, M. K., Yerima, S. Y., and Sezer, S. (2020). DL-droid: Deep learning based android malware detection using real devices. *Computers & Security*, 89:101663.

Arshad, S., Shah, M. A., Wahid, A., Mehmood, A., Song, H., and Yu, H. (2018). Samadroid: A novel 3-level hybrid malware detection model for android operating system. *IEEE Access*, 6:4321–4339.

Artenstein, N. and Revivo, I. (2014). Man in the binder: He who controls ipc, controls the droid. *BlackHat Europe 2014*.

Bridges, R. A., Glass-Vanderlan, T. R., Iannacone, M. D., Vincent, M. S., and Chen, Q. G. (2020). A survey of intrusion detection systems leveraging host data. *ACM Computing Surveys (CSUR)*, 52(6):1–35.

Google (2019). Monkey para testes de iu/apps.

Mohini, T., Kumar, S. A., and Nitesh, G. (2013). Review on android and smartphone security. *Research Journal of Computer and Information Technology Sciences*, 2320:6527.

Shen, D., Zhang, Z., Ding, X., Li, Z., and Deng, R. (2017). H-binder: A hardened binder framework on android systems. In *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 24–43. Springer International Publishing.

Yeon, J., Im, J., Jeon, B., Lee, S., Lee, E., and Bahn, H. (2016). Design and implementation of kernel binder cache to accelerate android ipc. In *2016 IEEE 22nd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 22–28.

Yerima, S. Y. and Sezer, S. (2019). Droidfusion: A novel multilevel classifier fusion approach for android malware detection. *IEEE Transactions on Cybernetics*, 49(2):453–466.

7. Apêndice

Este apêndice apresente todos os gráficos da curva ROC para os experimentos. A primeira figura representa os valores encontrados na Tabela 1, a segunda Figura representa

os valores da Tabela 2 e a terceira Figura representa os valores encontrados na Tabela 3 (o source das Figuras também esta presente no repositório do github).

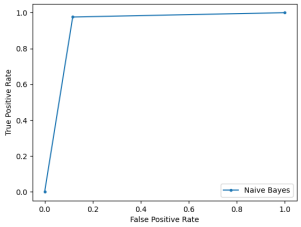

Figura 3. Naive Bayes sem k-fold.

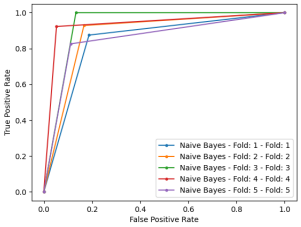

Figura 4. k-fold 80%.

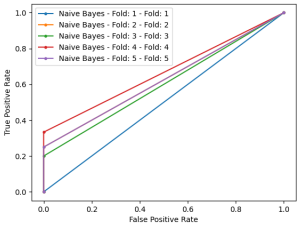

Figura 5. k-fold 20%.

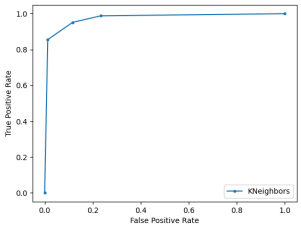

Figura 6. KNeighbors sem k-fold.

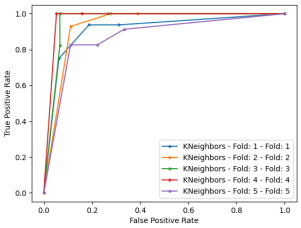

Figura 7. k-fold 80%.

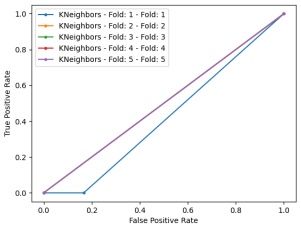

Figura 8. k-fold 20%.

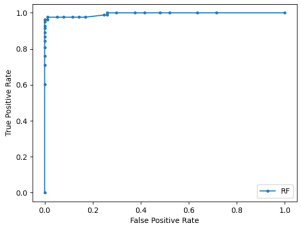

Figura 9. Random Forest sem k-fold.

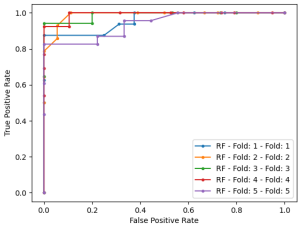

Figura 10. k-fold 80%.

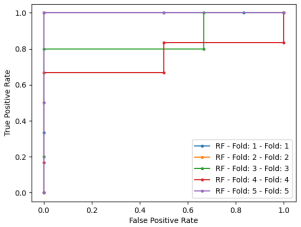

Figura 11. k-fold 20%.

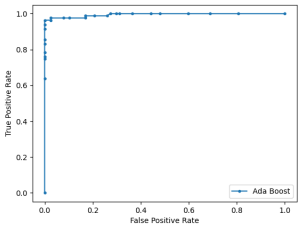

Figura 12. Ada Boost sem k-fold.

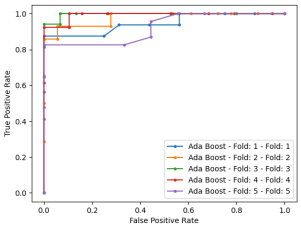

Figura 13. k-fold 80%.

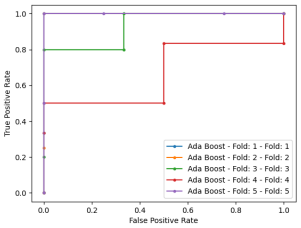

Figura 14. k-fold 20%.

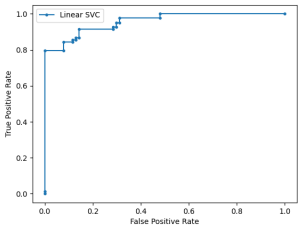

Figura 15. Linear SVC sem k-fold.

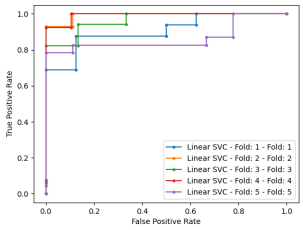

Figura 16. k-fold 80%.

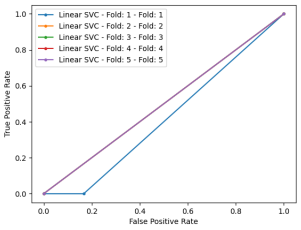

Figura 17. k-fold 20%.

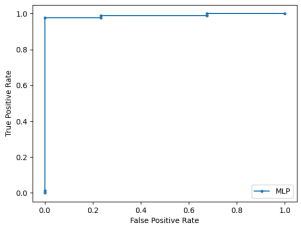

Figura 18. Multi-layer Perceptron sem k-fold.

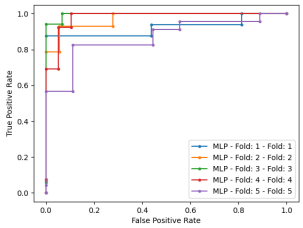

Figura 19. k-fold 80%.

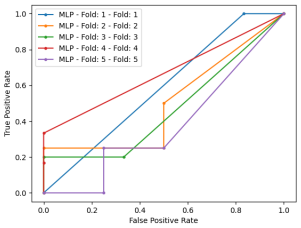

Figura 20. k-fold 20%.