Ataques de Mudança de Rótulo no Contexto da Detecção de *Malwares* Android: Uma Análise Experimental

Jonas Pontes[1,3], Eduardo Feitosa[1], Vanderson Rocha[1], Eduardo Souto[1], Diego Kreutz[2]

[1]Instituto de Computação – Universidade Federal do Amazonas (UFAM)
[2]Universidade Federal do Pampa (UNIPAMPA)
[3]Instituto Federal do Acre (IFAC)

{pontes,efeitosa,vanderson,esouto}@icomp.ufam.edu.br,
diegokreutz@unipampa.edu.br

Abstract. *In this article, we experimentally analyze seven datasets and three ML models in the context of three label flipping attacks and six classification noise rates for this. The results indicate that different adversarial algorithms can significantly degrade the models' performance and support the importance of developing defensive strategies to increase the security and effectiveness of ML models in the context of Android malware detection.*

Resumo. *Neste artigo, analisamos experimentalmente sete conjuntos de dados e três modelos de ML no contexto de três ataques de inversão de rótulos, organizados em seis taxas de ruído de classificação. Os resultados indicam que os diferentes algoritmos adversários de inversão de rótulos podem degradar significativamente o desempenho dos modelos e sustentam a importância de desenvolver estratégias defensivas para aumentar a segurança e a eficácia dos modelos de ML no contexto detecção de malwares Android.*

1. Introdução

No contexto de proteção de modelos contra ataques, o termo *"Adversarial Machine Learning"* (AML) refere-se a abordagens e técnicas desenvolvidas para fortalecer a robustez dos modelos de aprendizado de máquina (ML) contra ataques adversariais, nos quais exemplos de entrada sutilmente modificados são utilizados para enganar o modelo e obter resultados incorretos [Jmila and Khedher 2022]. Resumidamente, a defesa adversarial desempenha um papel fundamental na identificação de ataques maliciosos que são projetados para evadir os sistemas de detecção tradicionais [Jmila and Khedher 2022].

AML tem sido aplicada em diferentes domínios. Um bom exemplo é na filtragem de *spam*, onde a defesa adversarial identifica e bloqueia tentativas de envio projetadas para contornar os filtros convencionais [Zhang et al. 2021]. No contexto de detecção de *malwares*, a defesa adversarial identifica assinaturas ou comportamentos ocultos empregados para evadir a detecção em soluções convencionais [Taheri et al. 2020].

Os ataques de AML concentram-se principalmente em técnicas de envenenamento de dados (*Data Poisoning Attacks*) [Tabassi et al. 2019], cujo objetivo é comprometer a integridade de modelos de ML, ao modificar o conjunto de dados (*dataset*) utilizado durante a fase de treinamento do modelo. Ataques de envenenamento de dados direcionados e sofisticados podem ter como objetivo alterar pontos específicos nos dados, com o objetivo de evitar técnicas de defesa como a sanitização de dados [Koh et al. 2022].

Neste trabalho, apresentamos uma avaliação de ataques de envenenamento em *datasets* utilizados na detecção *malwares* Android. O foco é exclusivamente em ataques de inversão de rótulo (*Label Flipping Attacks*), também conhecidos como LFA. Esses ataques envolvem a manipulação intencional dos rótulos associados aos dados de treinamento, com o objetivo de comprometer a integridade dos modelos.

A seleção dos Ataques LFAs foi baseada no fato de que eles são adversários amplamente empregados em algoritmos de Aprendizado de Máquina [Yerlikaya and Şerif Bahtiyar 2022]. Adicionalmente, os LFAs têm a capacidade de introduzir ruídos na classificação por meio da manipulação exclusiva dos rótulos associados aos dados de treinamento, dispensando a exigência de acesso integral ao conjunto de dados alvo.

Esse é um tema especialmente sensível e crítico no contexto de detecção de *malwares* Android, onde os modelos são tipicamente treinados e validados a partir de *datasets* disponíveis publicamente em diferentes locais [Soares et al. 2021]. Muitas vezes são utilizadas cópias dos *datasets* originais, ou seja, há uma ampla redistribuição dos dados, o que potencializa ataques como os de inversão de rótulos. Adicionalmente, é importante destacar o fato de LFA ser executados com acesso limitado aos dados do *dataset*.

Para realizar a pesquisa, foram empregados três classificadores amplamente adotados na literatura: (*Support Vector Machine* - SVM, *Random Forest* - RF e *Decision Tree* - DT). Estudos recentes apontam que esses três classificadores estão entre os cinco mais empregados na detecção de *malware* Android [Sharma and Rattan 2021]. Além disso, sete *datasets* publicamente disponíveis e recorrentemente utilizados em trabalhos de detecção de *malwares* Android, como AndroCrawl, Adroit, DefenseDroid, Drebin-215, KronoDroid [Vilanova et al. 2021, Aurangzeb and Aleem 2023, Martín et al. 2016]. É importante destacar que todos os *datasets* consistem em características binárias, em que cada rótulo indica se uma amostra é benigna ou maligna. Por fim, os *datasets* e os classificadores são avaliados em relação a três estratégias de LFA: aleatória, baseada na entropia [Zhang et al. 2021] e *Silhouette Clustering* [Taheri et al. 2020]), inserindo ruídos (inversão de rótulos) que variam de 5% a 30%.

O restante do artigo está organizado da seguinte forma: A Seção 2 descreve os fundamentos de *adversarial machine learning* e dos ataques de envenenamento de dados. A Seção 3 explica a metodologia de experimentação, detalhando *datasets*, algoritmos, métricas e ambiente. Os resultados são apresentados na Seção 4. A Seção 5 apresenta uma discussão comparando trabalhos relacionados. As considerações finais e os trabalhos futuros são apresentados na Seção 6.

2. Adversarial Machine Learning (AML)

A área de AML é responsável por investigar as técnicas de ataques aos sistemas de ML, as consequências desses ataques e as estratégias de defesa correspondentes [Tabassi et al. 2019, Rosenberg et al. 2021]. Os ataques podem variar em relação aos alvos, às técnicas empregadas e ao conhecimento que o adversário possui sobre o sistema alvo. Além disso, esses ataques podem ocorrer tanto na fase de treinamento quanto na fase de teste do *pipeline* de ML [Tabassi et al. 2019].

Enquanto os ataques na etapa de treinamento buscam influenciar os dados de treinamento para gerar modelos que classifiquem erroneamente, os ataques na etapa de teste

têm o objetivo de comprometer o desempenho dos modelos [Tabassi et al. 2019]. Neste trabalho, concentramo-nos especificamente LFA, que atua na fase de treinamento do sistema de ML.

2.1. *Label Flipping Attack* (LFA)

LFA é um tipo especial de ataque de envenenamento de dados no qual um adversário busca comprometer a integridade dos modelos de AM, manipulando apenas os rótulos associados aos dados de treinamento [Papernot et al. 2016]. A motivação subjacente a esse tipo de ataque é explorar as vulnerabilidades presentes nos modelos de aprendizado de máquina, comprometendo a capacidade desses modelos de tomar decisões precisas e confiáveis ao desestruturar o processo de treinamento. Consequentemente, os ataques LFAs induzem o modelo a aprender a mapear os rótulos de maneira incorreta, seja de forma total ou parcial [Rosenfeld et al. 2020].

Formalmente, consideramos um LFA no contexto de classificação binária. Seja $D = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$ o subconjunto de treinamento do modelo de AM, onde n é a quantidade de amostras no subconjunto D. x_i representa os valores das características e $y_i \in \{0, 1\}$ é o rótulo correto correspondente à i-ésima amostra. No LFA, o adversário tem como objetivo alterar alguns dos rótulos verdadeiros y_i para rótulos $\hat{y}_i$, tal que $\hat{y}_i = 1 - y_i$. Essa alteração segue um algoritmo específico, que representa a estratégia de *flipping* adotada pelo adversário.

No contexto da detecção de *malwares* Android, o adversário busca associar características de aplicações maliciosas a rótulos benignos, ou vice-versa, levando o modelo a aprender relações incorretas entre as características de entrada e os rótulos associados. Isso pode levar a classificações errôneas durante o processo de detecção. Um LFA bem-sucedido pode comprometer a descoberta de *malwares*, permitindo que eles passem despercebidos e causem danos aos sistemas. Da mesma forma, classificar erroneamente aplicativos benignos como maliciosos pode causar inconvenientes aos usuários. A avaliação e identificação de classificadores e conjuntos de dados menos suscetíveis a esse tipo de ataque possibilitam decisões mais embasadas ao configurar um ambiente de detecção de malwares com base em técnicas de ML. Além disso, compreender a essência do problema conduzirá a futuras soluções para atenuar os efeitos dos Ataques LFAs.

É válido ressaltar que existem abordagens para mitigar os LFAs. Uma alternativa clássica e eficaz é a encriptação dos dados [Khalid et al. 2018], embora essa medida possa demandar recursos computacionais adicionais e não considera a possibilidade de rótulos originais incorretos. Outro exemplo de ideia é a utilização de higienização dos dados e o reforço da robustez dos algoritmos de aprendizado como contramedidas viáveis contra os LFAs durante a etapa de treinamento [Li et al. 2018].

3. Metodologia

A metodologia adotada neste estudo baseia-se em realizar a combinação de um dos sete *datasets* utilizados e um dos três modelos de ML, resultando em um total de 21 experimentos. Por exemplo, os experimentos 1, 2 e 3 compreendem a utilizado do *dataset* $AndroCrawl\_v1$ nos modelos SVM, RF e DT, respectivamente. O mesmo padrão é aplicado para os outros *datasets*. Essa abordagem possibilita que, para cada *dataset*, seja verificado qual classificador é mais adequado. Pela perspectiva do classificador, também permite obter o melhor *dataset* para treinar o modelo de classificação.

Cada um desses experimentos passará por 19 execuções. A primeira, denominada *baseline*, tem como objetivo obter os valores das métricas para cada classificador em cada *dataset*, sem a aplicação de qualquer técnica de inversão de rótulo. Partimos do pressuposto de que os *datasets* estão livres de contaminação e, assim, os resultados servem de base para avaliar o comportamento dos classificadores quando analisarmos dados contaminados nos experimentos posteriores. Nas 18 execuções seguintes, aplicamos obrigatoriamente uma das três técnicas de LFA adotadas, combinadas com taxas de ruído de 5%, 10%, 15%, 20%, 25% e 30%. As primeiras quatro taxas são baseadas no trabalho de [Zhang et al. 2021], enquanto as duas últimas foram adicionadas para ampliar os resultados obtidos.

3.1. Datasets

Neste trabalho, utilizamos sete *datasets* gerados a partir de cinco fontes de dados que contém informações de aplicações móveis Android, os quais podem ter características que denotam comportamentos potencialmente maliciosos. A seleção desses *datasets* teve, como requisito básico, ser de acesso livre a fim de garantir a reprodutibilidade dos resultados por outros pesquisadores.

Os *datasets* escolhidos são: (1) o $AndroCrawl\_v1$ e (2) o $AndroCrawl\_v2$, ambos derivados do AndroCrawl, em que o primeiro contém todas características binárias desse *dataset* e o segundo contém apenas características de permissões, chamadas de API e intenções; (3) *Adroit*[Martín et al. 2016], que utiliza dados de aplicativos disponíveis no Aptoide[1]; (4) *DefenseDroid*[Colaco et al. 2021], adotado em trabalhos recentes, como [Urooj et al. 2022] e [Rahali and Akhloufi 2023]; (5) *Drebin-215*, subconjunto do *dataset* Drebin [Arp et al. 2014] é o *dataset* mais utilizado em estudos sobre detecção de *malware* [Sharma and Rattan 2021]; (6) $KronoDroid\_real$ e (7) $KronoDroid\_emulator$, ambos derivados do *KronoDroid*[Guerra-Manzanares et al. 2021], possuem características extraídas de dispositivos reais e emulados, respectivamente.

A Tabela 1 apresenta um resumo das características e amostras de cada *dataset*, incluindo o número de amostras maliciosas e benignas, o total de amostras e o percentual de amostras duplicadas.

Tabela 1. Datasets

Dataset	Características	Amostras					
		Malwares		Benignos		Total	Duplicadas (%)
$AndroCrawl\_v1$	141	10170	10.5%	86562	89.5%	96732	37,08
$AndroCrawl\_v2$	81	10170	10.5%	86562	89.5%	96732	75,46
$Adroit$	166	3418	29.8%	8058	70.2%	11476	85,15
$DefenseDroid$	2938	6000	50.1%	5975	49.9%	11975	27,33
$Drebin-215$	215	5560	37%	9476	63%	15036	51,72
$KronoDroid\_real$	383	41382	53%	36755	47%	78137	56,08
$KronoDroid\_emulador$	383	28745	44.9%	35246	55.1%	63991	56,81

3.2. Algoritmos de LFA

Os três algoritmos utilizados neste trabalho são o LFA aleatório, LFA baseado em entropia [Zhang et al. 2021] e LFA baseado em *Silhouette Clustering* [Taheri et al. 2020]).

LFA Aleatório

No método LFA aleatório, desenvolvido especificamente para este trabalho, parte das instâncias referentes a *malware* no subconjunto de treinamento são reclassificadas como

[1] `https://en.aptoide.com/`

benignas. A partir dos rótulos desse subconjunto e da fração de rótulos que devem ser alterados, com valores possíveis de 0.05, 0.10, 0.15, 0.20, 0.25 e 0.30, é gerada uma lista de índices correspondente aos *malwares* do conjunto de treinamento e é determinado o número de instâncias que devem ser reclassificadas como benignas. Em seguida, de forma pseudoaleatória, é selecionada uma sublista contendo os índices que devem ser, de fato, modificados. Por fim, os rótulos das instâncias nos índices presentes na sublista são atualizados para 0 no conjunto de treinamento.

LFA baseado em Entropia

No LFA baseado em Entropia, a escolha das instâncias maliciosas do subconjunto de treinamento a serem reclassificadas como benignas é baseada nos valores dos pesos dessas instâncias, que são calculados utilizando a Entropia de Shannon [Shannon 2001]. Dado n, o número de instâncias do subconjunto de treinamento, e m, o número de características dessas instâncias, é calculada a proporção da i-ésima amostra sob o índice j (Equação 1). Em seguida, a entropia do j-ésimo atributo é dado pela Equação 2, onde $k = \frac{1}{ln(n)}$, de tal modo que $e_j > 0$. Finalmente, o peso do atributo j é calculado conforme a Equação 3.

$$p_{ij} = \frac{x_{ij}}{\sum_{i=1}^{n} x_{ij}}, 1 \leq i \leq n, 1 \leq j \leq m \quad (1)$$

$$e_j = -k \times \sum_{i=1}^{n} p_{ij} \times ln(p_{ij}) \quad (2)$$

$$W_j = \frac{1 - e_j}{m - \sum_{j=1}^{m} e_j} \quad (3)$$

Como base nos valores calculados utilizando a Equação 3, podemos implementar o algoritmo de inversão de rótulo baseado em entropia [Zhang et al. 2021], conforme apresentado no Algoritmo 1. Ao final do algoritmo, as amostras que possuem os menores valores são recategorizadas como aplicativos benignos. Em síntese, o algoritmo tem como objetivo maximizar a taxa de falsos negativos.

Algoritmo 1 - LFA baseado em Entropia.

Entrada: X_train, Y_train, f ▷ $f \rightarrow$ fração das instâncias maliciosas a serem reclassificadas
Saída: X_train, rótulos contaminados de Y_train

```
n ← tamanho(Y_train)
m ← quantidade de colunas de X_train
para i ← 1 até n faça
    se Y_train[i] = 1 então
        adicione(X_train[i], S)        ▷ adiciona X_train[i] a S, o subset de malwares
    fim se
fim para
para cada j ← 1 até m faça
    Compute W_j de acordo com a Equação 3        ▷ W → lista de pesos das características
fim para
para i ← 1 to n faça
    L_i ← Σ_{j=1}^{m} W_j × p_ij        ▷ L → lista de pesos de cada amostra maliciosa
fim para
S' ← ordene(S, "ascendente") de acordo com L
t ← f × tamanho(S)
Inverta os rótulos das primeiras t instâncias de S' em Y_train
```

LFA *Silhouette Clustering*

O método *Silhouette Clustering* [Taheri et al. 2020] propõe uma mudança tanto de instâncias malignas quanto benignas, tendo como entradas as características e rótulos do subconjunto de treinamento. A partir disso, ele constrói um modelo com dois *clusters* com o algoritmo K-Means, e usa-os para prever rótulos para as instâncias desse subconjunto.

Em seguida, são computadas as medidas do valor da silhueta a partir do subconjunto de treinamento e dos rótulos gerados, as quais são armazenadas em uma lista.

Na versão original desse algoritmo, as instâncias que possuem as medidas do valor da silhueta menores que zero têm o valor de seu rótulo modificado. Para permitir a utilização de frações fixas de instâncias a serem modificadas, como nas estratégias anteriores, a versão apresentada no Algoritmo 2 tem também como entrada a fração f de instâncias que se deseja recategorizar, a partir da qual – juntamente com o tamanho do subconjunto – a quantidade m de rótulos a serem modificados é calculada. Em seguida, os rótulos das instâncias que têm os m menores valores de silhueta são recategorizados.

Algoritmo 2 - LFA baseada no método Silhueta.

Entrada: X_train, Y_train, f ▷ $f \rightarrow$ fração das instâncias maliciosas a serem reclassificadas
Saída: rótulos contaminados de Y_train
  $M_K \leftarrow$ modelo com dois clusters construído usando o algoritmo KMeans
  $Rotulos \leftarrow$ prever rótulos de X_train usando M_K
  $S \leftarrow$ compute o valor da silhueta usando X_train e $Rotulos$
  $Indices \leftarrow$ índices de S ▷ Lista de índices de S, os mesmos de X_train
  $S' \leftarrow$ ordena(S, ´´ascendente")
  $Indices' \leftarrow$ índices de L ordenados de acordo com S'
  $m \leftarrow f \times tamanho(S)$
  $L \leftarrow$ primeiras m linhas de $Indices'$
  para cada $linha \in L$ **faça**
    $Y\_train[linha] \leftarrow valor\_absoluto(1 - Y\_train[linha])$
  fim para

3.3. Métricas

As métricas utilizadas para avaliar o desempenho dos modelos de ML neste trabalho são: *acurácia*, *precisão*, *recall* e *F1-Score* e *Matthews Correlation Coefficient* (MCC) [Chicco and Jurman 2020]. Apresentado na Equação 4, o MCC é uma métrica especialmente útil para classificação binária em *datasets* desbalanceados, isto é, quando uma classe está presente em uma proporção muito maior do que a outra, como é o caso dos *datasets* utilizados neste trabalho (ver Tabela 1). Por considerar as quatro predições possíveis para uma instância (verdadeiro positivo, verdadeiro negativo, falso positivo e falso negativo), ela é menos sensível a essa distribuição desigual das classes.

$$MCC = \frac{TP \times TN + FP \times FN}{\sqrt{((TP + FP) \times (TP + FN) \times (TN + FP) \times (TN + FN))}} \quad (4)$$

3.4. Configuração do Ambiente

Para a realizar os experimentos e suas execuções, utilizamos um computador com processador Intel Core i7-1185G7 oito núcleos de 3.0GHz da 11ª geração, com 32GB RAM e 512GB de HD. O sistema operacional nativo utilizado foi o Windows 11 Pro, com a opção de subsistema Linux para Windows ativada com o Ubuntu 22.04.1 LTS.

Quanto às implementações, foram utilizadas a linguagem Python, versão 3.10.6, e os modelos de AM presentes na biblioteca *scikit-learn*[2] sem quaisquer mudanças de configuração de parâmetros ou hiper-parâmetros. Para fins de completude, validação e reprodução, os *datasets* utilizados, as implementações dos algoritmos de LFA e os resultados gerais estão disponíveis no repositório GitHub público[3].

[2] https://scikit-learn.org/stable/

[3] https://github.com/Malware-Hunter/SBSeg23-flipping-attack

4. Resultados

Nesta seção apresentamos e discutimos os resultados das execuções *baselines* (Seção 4.1) e com LFA (Seção 4.2). Em todas as execuções foram utilizadas as cinco métricas, como demonstrado na seção 4.1. Contudo, devido à limitação de espaço, para as execuções com LFA, apenas a métrica MCC foi utilizada, uma vez que seus valores são mais ajustados para *datasets* desbalanceados [Chicco and Jurman 2020]. Os resultados que contemplam todas as métricas estão disponível no repositório do projeto.

4.1. Avaliação de desempenho dos baselines

Neste conjunto de execuções é analisado o desempenho dos classificadores em cada *dataset* original, sem a aplicação de inversão de rótulo. A Tabela 2 apresenta a relação de métricas para cada combinação de classificador e *dataset*.

Tabela 2. Desempenho dos modelos Support Vector Machine (SVM), Random Forest (RF), and Decision Tree (DT), usados como baseline.

Dataset	Classif.	Acurácia	Precisão	Recall	F1 Score	MCC
AndroCrawl_v1	SVM	98,3304	**94,5063**	89,3117	91,8356	0,9095
	RF	**98,3366**	94,4081	**89,4789**	**91,8774**	**0,9099**
	DT	97,6792	89,1049	88,7807	88,9425	0,8765
AndroCrawl_v2	SVM	**97,4341**	**95,8274**	79,0364	**86,6257**	**0,8570**
	RF	97,3111	92,2613	81,2389	86,4000	0,8513
	DT	96,9276	88,1169	**81,8092**	84,8460	0,8321
Adroit	SVM	**89,0728**	**87,1566**	74,2539	**80,1896**	**0,7316**
	RF	81,9014	66,3178	79,7250	72,4060	0,5966
	DT	80,3590	63,5349	**79,9298**	70,7955	0,5713
DefenseDroid	SVM	91,8914	**94,1218**	89,4000	91,7001	0,8389
	RF	**92,2756**	93,4727	**90,9333**	**92,1855**	**0,8458**
	DT	89,7453	90,6889	88,6333	89,6494	0,7951
Drebin-215	SVM	98,2642	98,7489	96,5288	97,6262	0,9627
	RF	**98,8893**	**99,3413**	**97,6439**	**98,4853**	**0,9762**
	DT	97,6523	96,5659	97,1043	96,8344	0,9497
KronoDroid_device	SVM	96,3014	97,1785	95,7977	96,4832	0,9259
	RF	**97,2817**	**97,8593**	**96,9890**	**97,4222**	**0,9455**
	DT	96,0121	96,2217	96,2496	96,2356	0,9200
KronoDroid_emulator	SVM	95,2118	95,2912	93,9850	94,6336	0,9032
	RF	**97,0590**	**97,0538**	**96,3785**	**96,7150**	**0,9405**
	DT	95,5525	94,9589	95,1505	95,0546	0,9101

Conforme apresentado na Tabela 2, o classificador RF apresenta os melhores resultados em cinco dos sete *datasets*, considerando as métricas acurácia, *recall*, *F1 Score* e MCC. Já o SVM destaca-se em *datasets* que possuem as mais altas porcentagens de amostras duplicadas. Por exemplo, ele possui desempenho superior em todas as métricas para $AndroCrawl\_v2$ e $Adroit$, coleções que possuem mais de três quartos das amostras duplicadas. O classificador DT, por sua vez, mostra os piores resultados para quase todas as métricas na maioria dos *datasets*, exceto nas métricas de *recall* nos *datasets* $AndroCrawl\_v2$ e $Adroit$. Nesses casos, seu desempenho é superior em razão dele ter categorizado menos *malwares* como benignos que os outros classificadores.

De maneira geral, os resultados indicam que os classificadores SVM e RF apresentam desempenhos satisfatórios e próximos em todas as métricas na maioria dos *datasets*. Também, nota-se que o comportamento dos classificadores varia significativamente entre os diferentes *datasets*. Isso enfatiza a importância de entender os dados, o problema e as compensações de cada modelo antes de escolher o modelo mais adequado.

4.2. Execuções com LFA

Flipping aleatório

A Tabela 3 apresenta os resultados da classificação, com a métrica de interesse sendo o MCC, após a mudança de rótulo de instâncias maliciosas. Os melhores resultados estão destacados em negrito.

Tabela 3. MCC para *flipping* aleatório

Dataset	Classif.	Ruído (%)						MCC baseline
		5	10	15	20	25	30	
AndroCrawl_v1	SVM	**0,9058**	**0,9038**	**0,8973**	**0,8970**	**0,8925**	**0,8897**	0,9095
	RF	0,9042	0,8978	0,8863	0,8756	0,8546	0,8312	0,9099
	DT	0,8397	0,8100	0,7848	0,7474	0,7328	0,6968	0,8765
AndroCrawl_v2	SVM	**0,8540**	**0,8525**	**0,8493**	**0,8428**	**0,8384**	**0,8329**	0,8570
	RF	0,8499	0,8415	0,8351	0,8200	0,7990	0,7852	0,8513
	DT	0,8094	0,7920	0,7720	0,7635	0,7382	0,7116	0,8321
Adroit	SVM	**0,7363**	**0,7294**	**0,7316**	**0,7261**	**0,7275**	0,7401	0,7316
	RF	0,5945	0,5916	0,6350	0,6082	0,6498	**0,7523**	0,5966
	DT	0,5760	0,6062	0,6869	0,5462	0,5708	0,5671	0,5713
DefenseDroid	SVM	0,8332	**0,8358**	**0,8338**	**0,8276**	**0,8187**	**0,7997**	0,8389
	RF	**0,8378**	0,8316	0,8225	0,8063	0,7914	0,7591	0,8458
	DT	0,7606	0,7262	0,7091	0,6652	0,6326	0,6103	0,7951
Drebin-215	SVM	0,9631	**0,9624**	**0,9592**	**0,9533**	**0,9446**	**0,9308**	0,9627
	RF	**0,9684**	0,9589	0,9473	0,9251	0,9055	0,8768	0,9762
	DT	0,9355	0,9114	0,8876	0,8707	0,8492	0,8275	0,9497
KronoDroid_device	SVM	0,9257	0,9261	**0,9254**	**0,9202**	**0,9099**	**0,8966**	0,9259
	RF	**0,9387**	**0,9287**	0,9163	0,8975	0,8759	0,8402	0,9455
	DT	0,8942	0,8658	0,8405	0,8124	0,7871	0,7523	0,9200
KronoDroid_emulator	SVM	0,8993	0,8953	0,8881	0,8831	**0,8750**	**0,8606**	0,9032
	RF	**0,9334**	**0,9257**	**0,9103**	**0,8920**	0,8676	0,8356	0,9405
	DT	0,8900	0,8682	0,8448	0,8226	0,8009	0,7653	0,9101

A inclusão de ruído nos *datasets* resultou em diferentes efeitos nos classificadores avaliados. Por exemplo, o classificador SVM demonstrou maior estabilidade em relação a inclusão de ruído. No $AndroCrawl\_v1$, seu MCC foi decaindo lentamente ao se aplicar os diferentes níveis de ruído, diminuindo de 0,9058 (ruído a 5%) para 0,8897 (ruído a 30%), uma queda máxima de 0,0198 ponto em relação do MCC no *baseline* (0,9095-0,8897). O mesmo acontece nos *datasets* $AndroCrawl\_v2$ (queda de 0,0241 ponto com 30% de ruído), $KronoDroid\_device$ (queda de 0,0266 ponto com 30% de ruído), $Drebin-215$ (queda de 0,0319 ponto com 30% de ruído), $DefenseDroid$ (queda de 0,0392 ponto com 30% de ruído) e $KronoDroid\_emulator$ (queda de 0,0426 ponto com 30% de ruído), apresentando uma queda média de 0,0307 ponto. Esse desempenho mais robusto do SVM pode ser atribuído à sua capacidade de encontrar hiperplanos ótimos de separação, o que o torna menos sensível a perturbações nos dados.

Por outro lado, o classificador RF apresentou quedas mais acentuadas no MCC à medida que o ruído aumenta, indicando maior suscetibilidade ao *flipping*. Por exemplo, no *baseline* ele classificou apenas 131 *malwares* como benigno no $Drebin-215$. Essa quantidade aumentou para 186 com 5% de ruído, chegando a 872 aplicativos maliciosos erroneamente classificados com 30% de ruído. Seu melhor resultado (resistência à inversão de rótulo) foi no *dataset* $AndroCrawl\_v1$, com queda de 0,0787 ponto para o ruído a 30%, e o pior no $KronoDroid\_emulator$, com queda de 0,1049 ponto para o ruído a 30%. Essa sensibilidade pode ser explicada pela natureza do RF, que combina várias árvores de decisão e pode ser influenciado por instâncias rotuladas incorretamente.

O classificador DT mostrou-se o mais suscetível à inversão dos rótulos, com quedas significativas no MCC em todos os *datasets*, exceto no $Adroit$. Em média, o DT teve uma diminuição de mais de 0,015 ponto no MCC ao aumentar o ruído para 30% nos *datasets*. Essa sensibilidade pode ser atribuída à construção de árvores de decisão, que são altamente dependentes da distribuição e representação dos dados.

Acerca dos *datasets*, a Figura 1 oferece uma visão geral dos valores de MCC para todos os classificadores nos diferentes *datasets* quando sujeitos à inclusão de ruído aleatoriamente.

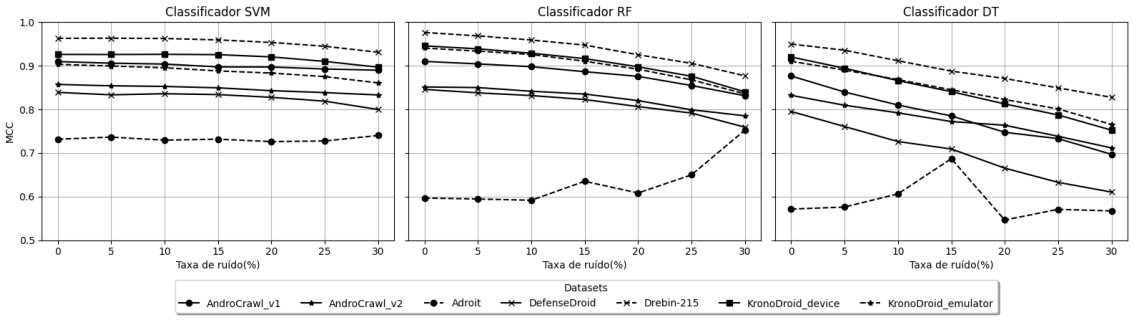

Figura 1. Efeito da inclusão de ruídos (método aleatório) nos *datasets*.

Nota-se que o $Adroit$ apresentou um comportamento diferente dos demais. Enquanto para outros a inclusão de ruídos provocou uma redução do MCC para os três classificadores, para o $Adroit$ esse valor aumenta. Como $Adroit$ é desbalanceado e com alta taxa de amostras duplicadas, esse aumento pode estar ligada à uma classificação que categoriza muitas instâncias como benignas, e o processo de inversão de rótulo torna-o ainda mais desbalanceado.

Os resultados também mostram que o $Drebin-215$ teve o menor decréscimo no MCC, em todos os níveis de ruídos, para os três classificadores, sugerindo que é o menos sensível à inclusão de ruído aleatório. Essa descoberta está em consonância com o trabalho de [Taheri et al. 2020], onde versões do $Drebin$ foram menos sensíveis à inclusão de ruído, embora utilizando uma estratégia diferente da aleatória.

Flipping baseado em entropia

A Tabela 4 apresenta os resultados da classificação com o método baseado na entropia e indica uma maior degradação ao desempenho dos modelos em comparação com o algoritmo aleatório. O classificador SVM mostrou-se mais adaptado às mudanças nos rótulos, mas teve decaimentos mais acentuados no MCC em comparação com o algoritmo aleatório. Por exemplo, nas execuções do $AndroCrawl\_v1$, o SVM teve uma queda no MCC de 0,9627 no *baseline* para 0,7537 com 30% de ruído. Essa diferença de 0,2090 ponto é significativamente maior do que a diferença observada com o algoritmo aleatório, que foi de 0,0319 ponto para as mesmas execuções. Comportamentos similares são observados nas demais execuções que envolvem o SVM, onde a diferença entre o *baseline* e o teste com 30% de ruído foi maior do que as diferenças com o algoritmo aleatório.

Esse comportamento é justificável pela própria na natureza dos dois algoritmos: enquanto o método aleatório escolhe qualquer instância maliciosa e reclassifica-a, o de entropia realiza a inversão baseado no menor peso, isto é, aquelas que tem mais semelhanças

Tabela 4. MCC a partir do algoritmo baseado em entropia

Dataset	Classif.	Ruído (%)						MCC baseline
		5	10	15	20	25	30	
AndroCrawl_v1	SVM	**0,8905**	**0,8639**	**0,8352**	**0,7989**	**0,7694**	**0,7333**	0,9095
	RF	0,8853	0,8546	0,8220	0,7895	0,7556	0,7186	0,9099
	DT	0,8497	0,8229	0,7926	0,7614	0,7312	0,6992	0,8765
AndroCrawl_v2	SVM	**0,8508**	**0,8209**	**0,7983**	**0,7720**	**0,7453**	**0,7179**	0,8570
	RF	0,8390	0,8164	0,7936	0,7623	0,7342	0,7061	0,8513
	DT	0,8201	0,7926	0,7665	0,7356	0,7099	0,6787	0,9321
Adroit	SVM	**0,7265**	**0,7276**	**0,7276**	**0,7276**	**0,7276**	**0,7276**	0,7316
	RF	0,5966	0,5981	0,5984	0,5983	0,5985	0,5983	0,5966
	DT	0,5387	0,5426	0,5415	0,5429	0,5426	0,5414	0,5713
DefenseDroid	SVM	0,8333	**0,8016**	**0,7663**	**0,7164**	**0,6719**	**0,6301**	0,8389
	RF	**0,8358**	0,7908	0,7430	0,7019	0,6571	0,6134	0,8458
	DT	0,7861	0,7343	0,6901	0,6545	0,6174	0,5921	0,7951
Drebin-215	SVM	0,9367	0,8954	**0,8528**	**0,8297**	**0,7888**	**0,7537**	0,9627
	RF	**0,9384**	**0,8999**	0,8522	0,8296	0,7877	0,7483	0,9762
	DT	0,9132	0,8746	0,8270	0,8040	0,7691	0,7287	0,9497
KronoDroid_device	SVM	0,8915	0,8448	0,8004	0,7625	0,7235	0,6791	0,9259
	RF	**0,9061**	**0,8586**	**0,8133**	**0,7723**	**0,7285**	**0,6861**	009455
	DT	0,8822	0,8387	0,7959	0,7541	0,7100	0,6682	0,9200
KronoDroid_emulator	SVM	0,8749	0,8359	0,7935	0,7557	0,7207	0,6925	0,9032
	RF	**0,8985**	**0,8637**	**0,8242**	**0,7833**	**0,7460**	**0,7106**	0,9405
	DT	0,8701	0,8390	0,7970	0,7603	0,7250	0,6917	0,9101

com instâncias benignas, o que potencializa a quantidade de falsos negativos. O classificador RF também mostrou-se mais suscetível a estratégia de inclusão de ruído baseada em entropia, com degradação em seu desempenho em todas as execuções e de maneira menos resiliente do que o SVM. Por exemplo, nas execuções desse classificador com o *dataset* $Drebin - 215$, o MCC diminuiu de 0,9762 no *baseline* para 0,7483 com 30% de ruído, uma diferença de 0,2279, que é consideravelmente maior do que a diferença de 0,0994 observada com o algoritmo aleatório para o mesmo *dataset*.

Por fim, de forma similar ao que ocorreu nas execuções com inversão de rótulo aleatoriamente, o classificador DT foi o mais sensível a inclusão de ruído de rotulação para todos os *datasets*. Para o nível de 30%, o maior valor do MCC observado foi 0,7653 (usando o $KronoDroid\_emulator$), ainda bem inferior a 0,8606 e 0,8356, valores observados nas execuções com os classificadores SVM e RF, respectivamente.

Sobre os *datasets*, a Figura 2 mostra os resultados.

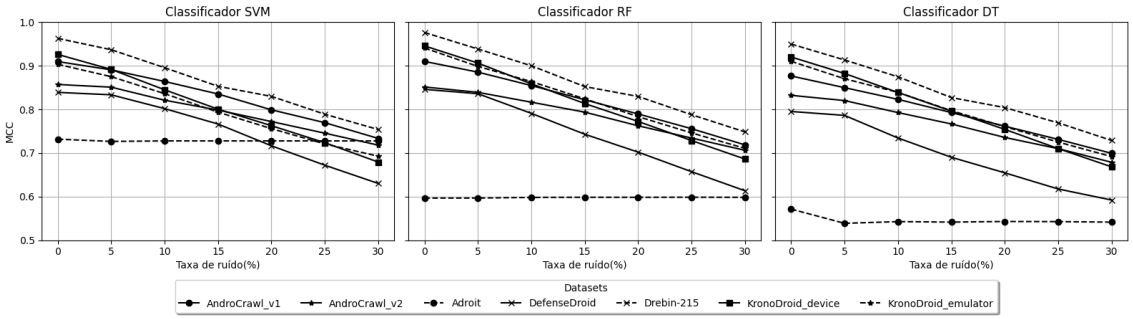

Figura 2. Efeito da inclusão de ruídos (método Entropia) nos *datasets*.

O $Drebin - 215$ apresenta as menores quedas de MCC para todos os classificadores, similar ao algoritmo de inclusão de ruído aleatório. O $Adroit$ demonstra flutuações mínimas no MCC, que não são discerníveis visualmente. Acreditamos que essa situação

decorre da presença de múltiplas amostras com comportamentos similares e, dependendo de onde o ataque ocorra, essas discrepâncias podem não impactar os classificadores de maneira substancial. Investigar esses aspectos será objeto de futuros trabalhos.

Flipping baseado no *Silhouette Clustering*

A Tabela 5 apresenta os resultados da classificação utilizando o algoritmo *Silhouette Clustering* com diferentes níveis de ruído.

Tabela 5. MCC a partir do método *Silhouette*.

Dataset	Classif.	Ruído (%)						MCC baseline
		5	10	15	20	25	30	
AndroCrawl_v1	SVM	**0,8445**	**0,6331**	0,4856	0,3811	0,2238	0,0543	0,9095
	RF	0,7688	0,6050	**0,4916**	**0,3893**	**0,2442**	**0,0743**	0,9099
	DT	0,6777	0,5566	0,4692	0,3705	0,2350	0,0675	0,8765
AndroCrawl_v2	SVM	**0,6747**	**0,5354**	**0,4459**	**0,3698**	**0,2831**	**0,1652**	0,8570
	RF	0,6713	0,5311	0,4380	0,3633	0,2667	0,1543	0,8513
	DT	0,6476	0,5170	0,4254	0,3551	0,2518	0,1479	0,8321
Adroit	SVM	**0,6472**	**0,6034**	**0,5555**	0,4812	0,4621	0,3668	0,7316
	RF	0,5295	0,4976	0,4874	**0,4957**	0,4593	0,4425	0,5989
	DT	0,5196	0,4869	0,5209	0,4926	**0,4670**	**0,4752**	0,5713
DefenseDroid	SVM	**0,8262**	0,7154	0,5831	0,4796	0,3934	0,3186	0,8389
	RF	0,8186	**0,7306**	**0,6098**	**0,5035**	**0,4024**	**0,3299**	0,8458
	DT	0,7095	0,6389	0,5447	0,4641	0,3840	0,2956	0,7951
Drebin-215	SVM	**0,9092**	**0,8167**	0,6721	0,5654	0,4780	0,3780	0,9627
	RF	0,8979	0,8087	**0,6934**	**0,5930**	**0,4972**	**0,3965**	0,9762
	DT	0,8469	0,7583	0,6670	0,5784	0,4846	0,3879	0,9497
KronoDroid_device	SVM	**0,8899**	0,7570	0,6749	0,5938	0,5151	0,4320	0,9255
	RF	0,8698	**0,7751**	**0,6900**	**0,6085**	**0,5285**	**0,4455**	0,9455
	DT	0,8273	0,7443	0,6661	0,5874	0,5047	0,4257	0,9200
KronoDroid_emulator	SVM	0,8223	0,7394	0,6506	0,5561	0,4593	0,3501	0,9032
	RF	**0,8536**	**0,7621**	**0,6680**	**0,5696**	**0,4755**	**0,3782**	0,9405
	DT	0,8218	0,7345	0,6462	0,5497	0,4641	0,3682	0,9101

Os resultados do método *Silhouette Clustering* indicam que o desempenho dos classificadores é comprometido à medida que o ruído aumenta. O fato do MCC se aproximar de zero significa que o modelo está tendo dificuldade em distinguir corretamente as classes e está realizando previsões ao acaso. Esse comportamento também é observado nos classificadores RF e DT, onde o MCC diminui à medida que o ruído aumenta.

Analisando os resultados por *dataset*, a Figura 3 destaca a capacidade de cada um em lidar com diferentes níveis de ruído.

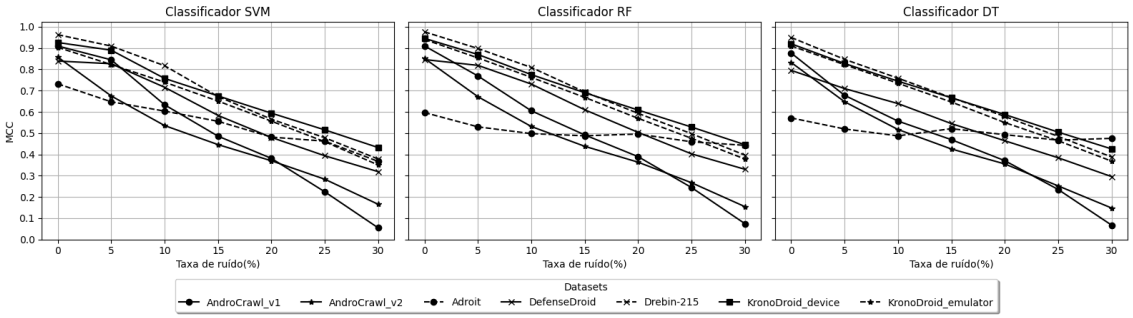

Figura 3. Efeito da inclusão de ruídos (método de *Silhouette*) nos *datasets*

.

Por exemplo, o $Drebin-215$ mostra uma melhor capacidade de lidar com níveis de ruído de até 15%, enquanto o $KronoDroid\_device$ é mais resiliente para níveis de

ruído de 20% e 25%. Já o $Adroit$ mostra uma maior tolerância ao nível de ruído de 30%. Essas observações indicam que a adição de ruído com base no método *Silhouette Clustering* pode afetar negativamente o desempenho dos classificadores, tornando a tarefa de classificação mais desafiadora.

5. Discussões

Para contextualizar e discutir nossos resultados, buscamos trabalhos na literatura que tenham aplicado ataques LFA no contexto de *malware* Android. No trabalho realizado por [Taheri et al. 2020], os autores utilizaram os *datasets* Android $Drebin$, $Contagio$[4] e $Genome$[5], divididos em *subsets* com características de chamadas API, permissões e intenções. Eles aplicaram o método Silhouette e testaram o desempenho dos classificadores SVM, DT, RF, *Neural Network* e *Convolutional Neural Networks* em todos os *datasets*. Observaram que os *subsets* do $Drebin$ foram os menos sensíveis ao ataque (a acurácia diminuiu de aproximadamente 98% para 83% nos cinco classificadores). No caso do Contagio, a diferença foi de 98% para 75%, aproximadamente. Já no $Genome$, a acurácia diminuiu de 99% para cerca de 72%.

Outro estudo relevante foi conduzido por [Bala et al. 2022], que utilizaram a estratégia de mudança aleatória de rótulos em amostras do $Drebin$ e uma versão modificada dele. Eles avaliaram a influência dessa estratégia de *flipping* nos modelos SVM. Os resultados destacaram que o desempenho do modelo de classificação diminuiu de 97.3% para 75.3% em precisão, de 97.3% para 74.7% em F1-Score e de 97.5% para 74.9% em *recall*.

Nosso trabalho complementa e estende, em parte, os estudos mencionados acima, considerando uma ampla variedade de *datasets* (sete), três dos modelos de ML mais usados na detecção de *malware*, três estratégias de LFA e seis taxas de ruído. Isso resulta em um conjunto abrangente de experimentos. Além disso, diferentemente dos estudos anteriores, consideramos o desbalanceamento presente nos *datasets* reais. Portanto, adotamos o MCC como a principal métrica de interesse, uma vez que é eficaz para avaliar modelos de ML em *datasets* desbalanceados.

Essas contribuições tornam nosso trabalho uma extensão significativa dos estudos existentes sobre ataques LFA em detecção de *malware* Android. Os resultados obtidos podem fornecer valiosas ideias sobre os desafios enfrentados pelos modelos de ML diante desses ataques e contribuir para criação de estratégias de defesa mais robustas.

6. Conclusões

Neste artigo, realizamos uma avaliação dos efeitos dos ataques de *Label Flipping Attacks* (LFA) na detecção de *malware* Android. Utilizamos diversos *datasets* binários para analisar o desempenho de três classificadores (SVM, RF e DT) com base na métrica MCC, que é mais adequada para *datasets* desbalanceados. Além disso, testamos três estratégias de inversão de rótulo e seis níveis de ruído predefinidos.

Nossos experimentos revelaram o impacto dos algoritmos de inversão de rótulo na classificação de *malwares* Android. Sob a perspectiva dos *datasets*, o algoritmo *Silhouette* mostrou-se mais eficaz em degradar o desempenho dos classificadores, em comparação com os algoritmos aleatório e Entropia. Em relação aos classificadores, constatamos que o

[4]http://contagiominidump.blogspot.com/

[5]http://www.malgenomeproject.org/

RF gerou os melhores detectores de *malware* a partir dos *datasets* sem ruído. No entanto, conforme incluímos ruídos na rotulação, o SVM demonstrou maior robustez. Para o algoritmo de inversão aleatória, a queda de desempenho desse classificador foi pequena.

Como trabalho futuro, pretendemos avaliar mais *datasets*; reavaliar o $Adroit$, que mostrou-se um *dataset* bastante enviesado; explorar outros classificadores e investigar novos algoritmos de LFA. Além disso, planejamos desenvolver contramedidas para mitigar os efeitos dos ataques de LFA em sistemas de detecção de *malware*.

Agradecimentos

Esta pesquisa foi parcialmente financiada, conforme previsto nos Arts. 21 e 22 do decreto no. 10.521/2020, nos termos da Lei Federal no. 8.387/1991, através do convênio no. 003/2021, firmado entre ICOMP/UFAM, Flextronics da Amazônia Ltda e Motorola Mobility Comércio de Produtos Eletrônicos Ltda. O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Código de Financiamento 001 e parcialmente financiado pela Fundação de Amparo à Pesquisa do Estado do Amazonas – FAPEAM – por meio do projeto POSGRAD.

Referências

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.

Aurangzeb, S. and Aleem, M. (2023). Evaluation and classification of obfuscated android malware through deep learning using ensemble voting mechanism. *Scientific Reports*, 13(1):3093.

Bala, N., Ahmar, A., Li, W., Tovar, F., Battu, A., and Bambarkar, P. (2022). Droidenemy: Battling adversarial example attacks for android malware detection. *Digital Communications and Networks*, 8(6):1040–1047.

Chicco, D. and Jurman, G. (2020). The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation. *BMC Genomics*, 21(1):6.

Colaco, C. W., Bagwe, M. D., Bose, S. A., and Jain, K. (2021). DefenseDroid: A Modern Approach to Android Malware Detection. *Strad Research*, 8(5):271–282.

Guerra-Manzanares, A., Bahsi, H., and Nõmm, S. (2021). Kronodroid: Time-based hybrid-featured dataset for effective android malware detection and characterization. *Computers & Security*, 110:102399.

Jmila, H. and Khedher, M. I. (2022). Adversarial machine learning for network intrusion detection: A comparative study. *Computer Networks*, 214:109073.

Khalid, F., Hanif, M. A., Rehman, S., and Shafique, M. (2018). Security for machine learning-based systems: Attacks and challenges during training and inference. In *2018 International Conference on Frontiers of Information Technology*, pages 327–332.

Koh, P. W., Steinhardt, J., and Liang, P. (2022). Stronger data poisoning attacks break data sanitization defenses. *Machine Learning*, 111(1):1–47.

Li, G., Zhu, P., Li, J., Yang, Z., Cao, N., and Chen, Z. (2018). Security matters: A survey on adversarial machine learning. *arXiv preprint arXiv:1810.07339*.

Martín, A., Calleja, A., Menéndez, H. D., Tapiador, J., and Camacho, D. (2016). Adroit: Android malware detection using meta-information. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–8.

Papernot, N., McDaniel, P., and Goodfellow, I. (2016). Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. *arXiv preprint arXiv:1605.07277*.

Rahali, A. and Akhloufi, M. A. (2023). Malbertv2: Code aware bert-based model for malware identification. *Big Data and Cognitive Computing*, 7(2):60.

Rosenberg, I., Shabtai, A., Elovici, Y., and Rokach, L. (2021). Adversarial machine learning attacks and defense methods in the cyber security domain. *ACM Comput. Surv.*, 54(5).

Rosenfeld, E., Winston, E., Ravikumar, P., and Kolter, Z. (2020). Certified robustness to label-flipping attacks via randomized smoothing. In *Proceedings of the 37th International Conference on Machine Learning*, pages 8230–8241. PMLR.

Shannon, C. E. (2001). A mathematical theory of communication. *SIGMOBILE Mob. Comput. Commun. Rev.*, 5(1):3–55.

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., Feitosa, E., and Kreutz, D. (2021). Detecção de malwares android: datasets e reprodutibilidade. In *Anais da XIX Escola Regional de Redes de Computadores*, pages 43–48. SBC.

Tabassi, E., Burns, K. J., Hadjimichael, M., Molina-Markham, A. D., and Sexton, J. T. (2019). A taxonomy and terminology of adversarial machine learning. *NIST IR*, 2019:1–29.

Taheri, R., Javidan, R., Shojafar, M., Pooranian, Z., Miri, A., and Conti, M. (2020). On defending against label flipping attacks on malware detection systems. *Neural Computing and Applications*, 32(18):14781–14800.

Urooj, B., Shah, M. A., Maple, C., Abbasi, M. K., and Riasat, S. (2022). Malware detection: A framework for reverse engineered android applications through machine learning algorithms. *IEEE Access*, 10:89031–89050.

Vilanova, L., Sayyed, R., Soares, T., Siqueira, G., Rodrigues, G., Feitosa, E., and Kreutz, D. (2021). Análise do impacto de viés nos conjuntos de dados para detecção de malwares android. In *Anais da XIX Escola Regional de Redes de Computadores*, pages 61–66. SBC.

Yerlikaya, F. A. and Şerif Bahtiyar (2022). Data poisoning attacks against machine learning algorithms. *Expert Systems with Applications*, 208:118101.

Zhang, H., Cheng, N., Zhang, Y., and Li, Z. (2021). Label flipping attacks against naive bayes on spam filtering systems. *Applied Intelligence*, 51(7):4503–4514.