# Impacto de ataques de evasão e eficácia da defesa baseada em treinamento adversário em detectores de malware

**Gabriel H. N. Espindola da Silva**[1], **Gilberto Fernandes Junior**[1], **e Bruno Bogaz Zarpelão**[1]

[1]Departamento de Computação – Universidade Estadual de Londrina (UEL)
Rod. Celso Garcia Cid, s/n – 86.057-970 – Londrina – PR – Brasil

{gabriel.henrique1, gilfernandes, brunozarpelao}@uel.br

***Abstract.*** *Machine learning (ML) algorithms can aid in detecting malicious software (malware), by identifying their behavior patterns. However, ML models are vulnerable to adversarial machine learning (AML) attacks, leading to malware being misclassified as benign. This study examines the impact of dFGSM (Deterministic Fast Gradient Sign Method), rFGSM (Randomic Fast Gradient Sign Method), BGA (Bit Gradient Ascent), BCA (Bit Coordinate Ascent), and Grosse attacks on malware detectors and evaluates the effectiveness of adversarial training as a defense. The results demonstrate that high-intensity attacks (dFGSM, rFGSM, BGA) significantly reduce the detector accuracy, even when adversarial training is applied.*

***Resumo.*** *Algoritmos de aprendizado de máquina (AM) podem ajudar na detecção de programas maliciosos, ou malwares, ao identificar padrões de comportamento deles. No entanto, os modelos de AM são vulneráveis a ataques de aprendizado de máquina adversário (AMA), permitindo que malwares sejam classificados como benignos. Este trabalho investiga o impacto dos ataques dFGSM (Deterministic Fast Gradient Sign Method), rFGSM (Randomic Fast Gradient Sign Method), BGA (Bit Gradient Ascent), BCA (Bit Coordinate Ascent) e Grosse contra detectores de malware e a eficácia da defesa baseada em treinamento adversário. Os resultados mostraram que ataques de alta intensidade (dFGSM, rFGSM, BGA) impactam significativamente a acurácia dos detectores, mesmo com treinamento adversário.*

## 1. Introdução

Os programas maliciosos (*malware*) frequentemente causam incidentes de segurança, escondendo suas intenções por meio de técnicas que enganam usuários e sistemas de segurança. Para combatê-los, detectores de malware utilizam técnicas variadas, mas a capacidade dos malwares de se disfarçarem como programas benignos torna essa tarefa desafiadora, levando a falsos negativos. O uso de técnicas de aprendizado de máquina tornou-se um reforço importante, permitindo a análise de padrões em malwares por meio de análises estáticas e dinâmicas e a construção de bancos de dados para auxiliar na classificação de programas. No entanto, essa abordagem também é vulnerável a ataques com amostras adversárias, que exploram brechas nos modelos de AM para subverter a detecção, fazendo com que malwares sejam classificados como benignos sem perder suas propriedades nocivas [Aslan and Samet 2020, Papernot et al. 2017, Kolosnjaji et al. 2018, Grosse et al. 2016].

Diante disso, esse trabalho investiga diferentes ataques contra um conjunto de detectores de malware destinados a aplicativos Android. O experimento inclui, primeiramente, um conjunto de modelos alvo, composto por redes neurais com topologias diferentes e uma Random Forest. Adicionalmente, cinco dessas redes neurais são retreinadas utilizando o treinamento adversário. O experimento também conta com um conjunto de modelos de ataque, a partir dos quais são geradas amostras adversárias utilizando as técnicas dFGSM (*Deterministic Fast Gradient Sign Method*), rFGSM (*Randomic Fast Gradient Sign Method*), BGA (*Bit Gradient Ascent*), BCA (*Bit Coordinate Ascent*) e Grosse. Observou-se que ataques de alta intensidade, tais como dFGSM, rFGSM e BGA, apresentam impacto significativo contra as redes neurais, tanto com defesa quanto sem defesa, bem como contra a Random Forest. Ataques de baixa intensidade como o BCA e Grosse foram efetivos apenas contra as redes neurais sem defesa. Além disso, observamos que a variação na topologia dos modelos das redes neurais não teve um impacto significativo na geração de ataques ou na robustez dos modelos.

O restante deste artigo é organizado da seguinte forma. A Seção 2 traz os trabalhos relacionados sobre detecção de malware e amostras adversárias. Na Seção 3, discutimos os experimentos desenvolvidos. A Seção 4 apresenta os resultados dos experimentos, enquanto a Seção 5 encerra o artigo com as considerações finais.

## 2. Trabalhos Relacionados

Os estudos sobre aprendizado de máquina adversário começaram no campo da visão computacional e posteriormente se expandiram para outras áreas, como a detecção de malwares. Grosse et al. [2017] propuseram um ataque baseado no cálculo da matriz Jacobiana, adaptando uma ideia de visão computacional para detectar malwares, enquanto Rosenberg et al. [2018] introduziram um ataque contra classificadores de malware utilizando o conceito de *transferability*, gerando amostras adversárias para atributos estáticos e dinâmicos. Pierazzi et al. [2020] propuseram uma técnica de camuflagem de malwares, que insere códigos benignos em malwares, tornando-os mais difíceis de detectar, mesmo por modelos robustos como o Sec-SVM. Al-Dujaili et al. [2018] apresentaram dois novos ataques, BGA e BCA, comparando-os com métodos existentes, enquanto Shaukat et al. [2022] desenvolveram um método para aumentar a robustez dos detectores de malware por meio de treinamento adversário. Os estudos de Shaukat et al., Al-Dujaili et al., e Rosenberg et al. investigam detectores de malware específicos para o sistema operacional Windows, enquanto Grosse et al. focam em aplicativos Android.

Este trabalho se concentra em aplicativos Android, comparando o desempenho de redes neurais, com e sem treinamento adversário, com algoritmos mais robustos como Random Forest, além de investigar se variações nas topologias das redes tornam os modelos menos suscetíveis a amostras maliciosas.

## 3. Materiais e Métodos

O experimento proposto neste trabalho busca responder as seguintes perguntas:

1. Qual a severidade dos ataques, considerando que eles adotam estratégias distintas para preparar as amostras adversárias?
2. Qual a eficácia do mecanismo de defesa baseado em treinamento adversário?
3. Qual a influência da topologia do modelo de rede neural no ataque e defesa?

4. Qual o desempenho da Random Forest comparado às redes neurais, já que todos os tipos de ataque experimentados são voltados para redes neurais?

   As próximas seções mostram os detalhes do experimento.

### 3.1. Conjunto de dados

O conjunto de dados é voltado para o sistema operacional Android, tendo sido criado e fornecido pelo projeto Drebin[1] [Arp et al. 2014]. Para a extração dos atributos, foi realizada uma análise estática sobre os aplicativos, considerando apenas seus manifestos e códigos dex. A partir da análise, os atributos extraídos foram organizados em 8 grupos, cada um deles representando um tipo de informação. Quatro grupos foram extraídos do manifesto: elementos de hardware, permissões, componentes, e filtros de *intents*. Outros quatro grupos foram extraídos do código dex: chamadas de API restritas, permissões utilizadas, chamadas de API suspeitas, e endereços de rede. Os atributos extraídos de ambos os arquivos podem ser entendidos como características de um aplicativo, que descrevem como ele opera. Para cada aplicativo, será construído um vetor de atributos. Caso o aplicativo possua uma característica, a respectiva posição em seu vetor de atributos será marcada como 1. Em caso contrário, a posição do vetor é marcada com o valor 0.

### 3.2. Características do adversário

Para alterar os malwares de forma que eles não sejam detectados, o atacante executa os seguintes passos. O primeiro passo é a criação de um conjunto de dados, onde o atacante precisa obter aplicativos benignos e maliciosos, utilizando, por exemplo, sites como o VirusShare[2]. Com o conjunto de dados em mãos, o atacante extrai os atributos e treina uma rede neural que simula o detector alvo. Em seguida, o atacante gera uma versão modificada do seu malware. Utilizando a rede neural treinada, o atacante explora vulnerabilidades do modelo, identifica os atributos que causam evasão da classificação, e modifica o malware para que seja classificado como benigno. Finalmente, o atacante propaga a versão modificada do malware, visando um sistema específico ou vários sistemas indiscriminadamente, com a expectativa de que alguns sistemas de detecção falhem e permitam que o malware atue como se fosse um aplicativo benigno.

### 3.3. Treinamento dos modelos

O treinamento das redes neurais e da Random Forest seguirá o mesmo procedimento. Durante a fase de treinamento, os dados terão uma proporção equilibrada de $50\%$ de amostras maliciosas e benignas. Para o treinamento, utilizaremos um total de 4.000 amostras selecionadas aleatoriamente. Já na fase de inferência, utilizaremos a proporção de 24:1 (presente no conjunto de dados). São 2.400 amostras benignas e 100 maliciosas, totalizando 2.500 amostras. Portanto, a proporção de treinamento e teste ficará em torno de 60:40. A quantidade de atributos será determinada a partir dos aplicativos presentes no conjunto de dados de treinamento. Considerando as 4.000 amostras selecionadas, temos 25.550 atributos.

As redes neurais são construídas utilizando a biblioteca *Pytorch*[3] na versão $2.0$. Para cada neurônio nas camadas ocultas, atribuímos a função de ativação ReLu (*rectifier*

[1]Disponível em: https://www.sec.tu-bs.de/ danarp/drebin/index.html
[2]Disponível em: https://virusshare.com/
[3]Disponível em: https://pytorch.org/

*linear unit*). Na camada de saída do modelo, utilizamos a função de ativação LogSoftmax. A função de custo dos modelos é a *CrossEntropyLoss*, e utilizamos o algoritmo Adam para otimizá-la. Cada *batch* contém 200 amostras, sendo $50\%$ delas benignas e $50\%$ maliciosas. O modelo passa pelos dados de treinamento 200 vezes (*epochs*) com uma taxa de aprendizado de 0,001. A implementação da Random Forest foi feita utilizando a biblioteca *Scikit-learn*[4] [Pedregosa et al. 2011]. O algoritmo é composto de 100 árvores e utilizamos o critério Gini para medir a qualidade das divisões. Utilizamos a opção de *bootstrap* como verdadeira, permitindo que o algoritmo crie amostras aleatórias na construção das árvores.

### 3.4. Métodos para geração de amostras adversárias

Todos os métodos utilizados para gerar amostras adversárias, rFGSM, dFGSM, BGA, BCA e Grosse, são baseados em um modelo de classificação pré-treinado. O dFGSM e rFGSM são variações do FGSM, que utiliza o gradiente da função de erro do modelo para perturbar a entrada e maximizar a alteração na saída. O dFGSM aplica um limiar determinístico para modificar atributos, enquanto o rFGSM usa um limiar randômico. O método de Grosse modifica $k$ atributos com base na matriz Jacobiana para maximizar o erro do modelo. Os ataques BGA e BCA são específicos para malwares. O BGA modifica o atributo cuja derivada parcial contribua mais ou igualmente para a norma $\ell_2$ (distância Euclidiana) do gradiente em comparação com o restante dos atributos, e o BCA, similar ao método de Grosse, escolhe o atributo que maximiza o erro do modelo em cada iteração. Para os ataques Grosse e BCA, escolhemos $k = 20$. Para os ataques dFGSM, rFGSM e BGA, utilizamos $k = 50$.

### 3.5. Defesa baseada em treinamento adversário

O treinamento adversário é simples: durante a fase de treinamento do modelo alvo, são apresentadas amostras maliciosas ao modelo para que ele aprenda a se proteger contra esses ataques. Neste estudo, selecionamos aleatoriamente $200$ malwares que não foram usados no treinamento ou inferência. Para cada ataque utilizado, geramos 200 amostras adversárias a partir desses dados, totalizando 1.000 novas amostras. Em seguida, retreinamos o modelo com essas amostras adversárias e 1.000 amostras benignas, mantendo o equilíbrio das classes. No total, o treinamento conta com 6.000 amostras: as 4.000 iniciais, mais 1.000 amostras adversárias e 1.000 benignas.

### 3.6. Métricas de avaliação

As métricas de precisão, sensibilidade e *F1-score* são utilizadas para avaliar a capacidade preditiva dos modelos. A precisão ($prec = \frac{VP}{VP+FP}$) observa a quantidade de falsos positivos em relação a todas amostras classificadas como positivas. A sensibilidade ($sens = \frac{VP}{VP+FN}$) mede a proporção de acertos ao classificar amostras positivas. Já o F1-score ($f1 = 2 * \frac{prec*sens}{prec+sens}$) é uma média harmônica entre a precisão e sensibilidade. $VN$, $VP$, $FN$, e $FP$ significam, respectivamente, verdadeiro negativo, verdadeiro positivo, falso negativo, e falso positivo. Como o objetivo do detector de malware é minimizar a quantidade de falsos negativos, e os ataques almejam o contrário, utilizaremos uma métrica baseada na observação da sensibilidade para medir a severidade do ataque ($SA$) contra o modelo. Ela estabelece uma razão entre a sensibilidade antes ($sens_{antes}$) e depois do ataque ($sens_{depois}$): $SA = 1 - \frac{sens_{depois}}{sens_{antes}}$.

[4]Disponível em: https://scikit-learn.org/stable/index.html

| Nome | Modelo | Sensibilidade | Precisão | F1-Score |
|---|---|---|---|---|
| $M_1$ | 5000x1000x500x200 | 0,910 | 0,497 | 0,643 |
| $M_2$ | 5000x200 | 0,940 | 0,482 | 0,637 |
| $M_3$ | 1000x1000x1000 | 0,940 | 0,472 | 0,629 |
| $M_4$ | 200x500x200 | 0,930 | 0,451 | 0,608 |
| $M_5$ | 200x200x200 | 0,940 | 0,433 | 0,593 |

**Tabela 1. Modelos de rede neural escolhidos para o experimento.**

| Ataques contra modelos sem defesa | | | | | |
|---|---|---|---|---|---|
| | $M_1$ | $M_2$ | $M_3$ | $M_4$ | $M_5$ |
| $M_1$-$M_5$ **x dFGSM** | 0,8498 $\pm$ 0,0021 | 1,0000 $\pm$ 0,0000 | 0,9463 $\pm$ 0,0008 | 1,0000 $\pm$ 0,0000 | 1,0000 $\pm$ 0,0000 |
| $M_1$-$M_5$ **x rFGSM** | 0,8498 $\pm$ 0,0021 | 1,0000 $\pm$ 0,0000 | 0,9463 $\pm$ 0,0008 | 1,0000 $\pm$ 0,0000 | 1,0000 $\pm$ 0,0000 |
| $M_1$-$M_5$ **x BGA** | 1,0000 $\pm$ 0.0000 | 1,0000 $\pm$ 0,0000 | 1,0000 $\pm$ 0,0000 | 1,0000 $\pm$ 0,0000 | 1,0000 $\pm$ 0,0000 |
| $M_1$-$M_5$ **x BCA** | 0,6028 $\pm$ 0,0236 | 0,6480 $\pm$ 0,0133 | 0,9439 $\pm$ 0,0108 | 0,5879 $\pm$ 0,0236 | 0,7146 $\pm$ 0,0052 |
| $M_1$-$M_5$ **x Grosse** | 0,9743 $\pm$ 0,0193 | 0,9699 $\pm$ 0,0191 | 0,9699 $\pm$ 0,0049 | 0,9786 $\pm$ 0,0168 | 0,9592 $\pm$ 0,0275 |

**Tabela 2. Severidade média dos ataques contra os 5 modelos sem defesa. As linhas representam a média e o desvio padrão da severidade do ataque contra os 5 modelos e as colunas mostram qual a topologia utilizada pelo atacante.**

## 4. Resultados

O primeiro passo do experimento é selecionar 5 modelos de redes neurais, tendo em conta o *F1-score* deles. Foram criados 30 modelos com topologias diferentes, variando a quantidade de camadas e neurônios por camada, e apenas os que obtiveram os maiores *F1-scores* foram escolhidos. A Tabela 1 apresenta os 5 modelos escolhidos para os cenários.

### 4.1. Desempenho das redes neurais nos cenários de ataque e defesa

As Tabelas 2 e 3 mostram a severidade média dos ataques contra os modelos utilizados neste trabalho. Ambas as tabelas foram construídas estabelecendo uma média da severidade dos ataques contra cada topologia. A diferença na topologia dos modelos não proporcionou uma maior resistência aos ataques nem possibilitou ataques mais efetivos. Em média, os ataques alcançaram impactos consideráveis contra os modelos. Observando a Tabela 2, vemos os modelos quando não possuem defesa. Os ataques dFGSM, rFGSM e BGA obtiveram severidades máximas ($severidade = 1$) contra todos os modelos alvos em pelo menos uma situação. O método de Grosse teve um desempenho muito próximo dos ataques mencionados acima. O ataque que obteve menos sucesso nesse cenário foi o BCA. Na Tabela 3, onde os modelos alvo receberam treinamento adversário, observamos uma diminuição aguda da severidade do ataque com os métodos BCA e Grosse. Por outro lado, os ataques dFGSM, rFGSM e BGA continuaram apresentando uma elevada severidade, atingindo novamente valores máximos.

### 4.2. Comparação entre a Random Forest e as redes neurais

Os resultados para a Random Forest (Tabela 4) foram comparáveis aos das redes neurais. Em geral, os ataques de alta intensidade tiveram impactos significativos contra a Random Forest. Já os métodos BCA e Grosse tiveram severidade igual a zero para todos os modelos utilizados pelos atacantes. Mesmo sem o treinamento adversário, a Random Forest apresentou uma robustez considerável contra os ataques de baixa intensidade. Parte disso pode ser explicado pela forma como o algoritmo funciona. A Random Forest emprega

| **Ataques contra modelos com defesa** | | | | | |
|---|---|---|---|---|---|
| | $M_1$ | $M_2$ | $M_3$ | $M_4$ | $M_5$ |
| $M_1$-$M_5$ **x dFGSM** | 0,8453 $\pm$ 0,0043 | 1,0000 $\pm$ 0,0000 | 0,9439 $\pm$ 0,0014 | 1,0000 $\pm$ 0,0000 | 0,8000 $\pm$ 0,4472 |
| $M_1$-$M_5$ **x rFGSM** | 0,8453 $\pm$ 0,0043 | 1,0000 $\pm$ 0,0000 | 0,9439 $\pm$ 0,0014 | 1,0000 $\pm$ 0,0000 | 0,8000 $\pm$ 0,4472 |
| $M_1$-$M_5$ **x BGA** | 0,9682 $\pm$ 0,0711 | 1,0000 $\pm$ 0,0000 | 0,9886 $\pm$ 0,0254 | 1,0000 $\pm$ 0,0000 | 1,0000 $\pm$ 0.0000 |
| $M_1$-$M_5$ **x BCA** | 0,3157 $\pm$ 0,1360 | 0,5416 $\pm$ 0,1289 | 0,2870 $\pm$ 0,0855 | 0,4251 $\pm$ 0,1258 | 0,2610 $\pm$ 0,2675 |
| $M_1$-$M_5$ **x Grosse** | 0,0222 $\pm$ 0,0281 | 0,2181 $\pm$ 0,1496 | 0,0666 $\pm$ 0,0662 | 0,1304 $\pm$ 0,0889 | 0,0273 $\pm$ 0,0499 |

**Tabela 3. Severidade média dos ataques contra os 5 modelos com defesas. As linhas representam a média e o desvio padrão da severidade do ataque contra os 5 modelos e as colunas mostram qual a topologia utilizada pelo atacante.**

| **Ataques contra Random Forest sem defesa** | | | | | |
|---|---|---|---|---|---|
| | **M1** | **M2** | **M3** | **M4** | **M5** |
| **RF x dFGSM** | 0,848 | 1,000 | 0,946 | 1,000 | 0,870 |
| **RF x rFGSM** | 0,761 | 0,891 | 0,870 | 0,902 | 0,511 |
| **RF x BGA** | 0,989 | 1,000 | 0,978 | 1,000 | 0,957 |
| **RF x BCA** | 0,000 | 0,000 | 0,000 | 0,000 | 0,000 |
| **RF x Grosse** | 0,000 | 0,000 | 0,000 | 0,000 | 0,000 |

**Tabela 4. Severidade dos ataques contra a Random Forest. As linhas representam a severidade do ataque contra a Random Forest e as colunas mostram qual a topologia utilizada pelo atacante.**

a técnica de *feature bagging*, a qual consiste na criação de conjuntos de atributos distintos para cada árvore utilizada no processo de *ensemble*. Essa estratégia permite que cada árvore seja treinada com um subconjunto de características. Durante esse processo, os atributos manipulados pelos ataques de baixa intensidade são dissipados em várias árvores ou poucas árvores acabam contendo atributos que são manipulados. Ao observamos os ataques de alta intensidade, temos que o mesmo processo ocorre, mas dessa vez, por conta da grande quantidade de atributos manipulados, os atributos maliciosos estão presentes em mais árvores. Isso possibilita que mais árvores sejam atacadas, convergindo para um ataque mais severo.

## 5. Conclusão

Com o objetivo de contribuir para os estudos na área de aprendizado de máquina aplicado à classificação de malwares, este trabalho explorou o impacto dos ataques adversários contra detectores de malwares. Foi observado que os detectores são menos suscetíveis a ataques de baixa intensidade, como é o caso do BCA e Grosse. Entretanto, contra ataques como dFGSM, rFGSM e BGA, ataques de maior intensidade, os classificadores não se mostraram robustos, nem mesmo quando utilizando o treinamento adversário. Apesar disso, ataques de alta intensidade são mais fáceis de serem detectados durante fases que precedem a classificação das amostras. Além disso, não foi possível determinar se redes neurais mais complexas apresentam uma maior robustez contra exemplos adversários. No que tange diferentes algoritmos contra as amostras adversárias, a Random Forest se mostrou bastante promissora. Mesmo não utilizando o treinamento adversário, ela conseguiu anular os ataques de baixa intensidade, se mostrando um algoritmo mais robusto que as redes neurais utilizadas neste experimento.

## Referências

Al-Dujaili, A., Huang, A., Hemberg, E., and O'Reilly, U.-M. (2018). Adversarial deep learning for robust detection of binary encoded malware. In *2018 IEEE Security and Privacy Workshops (SPW)*, pages 76–82. IEEE.

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.

Aslan, Ö. A. and Samet, R. (2020). A comprehensive review on malware detection approaches. *IEEE Access*, 8:6249–6271.

Grosse, K., Papernot, N., Manoharan, P., Backes, M., and McDaniel, P. (2016). Adversarial perturbations against deep neural networks for malware classification. *arXiv preprint arXiv:1606.04435*.

Grosse, K., Papernot, N., Manoharan, P., Backes, M., and McDaniel, P. (2017). Adversarial examples for malware detection. In *European symposium on research in computer security*, pages 62–79. Springer.

Kolosnjaji, B., Demontis, A., Biggio, B., Maiorca, D., Giacinto, G., Eckert, C., and Roli, F. (2018). Adversarial malware binaries: Evading deep learning for malware detection in executables. In *2018 26th European signal processing conference (EUSIPCO)*, pages 533–537. IEEE.

Papernot, N., McDaniel, P., Goodfellow, I., Jha, S., Celik, Z. B., and Swami, A. (2017). Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, pages 506–519.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Pierazzi, F., Pendlebury, F., Cortellazzi, J., and Cavallaro, L. (2020). Intriguing properties of adversarial ml attacks in the problem space. In *2020 IEEE symposium on security and privacy (SP)*, pages 1332–1349. IEEE.

Rosenberg, I., Shabtai, A., Rokach, L., and Elovici, Y. (2018). Generic black-box end-to-end attack against state of the art api call based malware classifiers. In *International Symposium on Research in Attacks, Intrusions, and Defenses*, pages 490–510. Springer.

Shaukat, K., Luo, S., and Varadharajan, V. (2022). A novel method for improving the robustness of deep learning-based malware detectors against adversarial attacks. *Engineering Applications of Artificial Intelligence*, 116:105461.