Universidade Estadual de Londrina

GABRIEL H. N. E. SILVA

APRENDIZADO DE MÁQUINA ADVERSÁRIO CONTRA DETECTORES DE MALWARE

LONDRINA
2023

GABRIEL H. N. E. SILVA

APRENDIZADO DE MÁQUINA ADVERSÁRIO CONTRA DETECTORES DE MALWARE

Trabalho de Conclusão de Curso apresentado ao Curso de Bacharelado em Ciência da Computação do Departamento de Computação da Universidade Estadual de Londrina, como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Bruno Bogaz Zarpelão

LONDRINA
2023

Ficha de identificação da obra elaborada pelo autor, através do Programa de Geração Automática do Sistema de Bibliotecas da UEL

Sobrenome, Nome.
Título do Trabalho : Subtitulo do Trabalho / Nome Sobrenome. - Londrina, 2017.
100 f. : il.

Orientador: Nome do Orientador Sobrenome do Orientador.
Coorientador: Nome Coorientador Sobrenome Coorientador.
Dissertação (Mestrado em Ciência da Computação) - Universidade Estadual de Londrina, Centro de Ciências Exatas, Programa de Pós-Graduação em Ciência da Computação, 2017.
Inclui bibliografia.

1. Assunto 1 - Tese. 2. Assunto 2 - Tese. 3. Assunto 3 - Tese. 4. Assunto 4 - Tese. I. Sobrenome do Orientador, Nome do Orientador. II. Sobrenome Coorientador, Nome Coorientador. III. Universidade Estadual de Londrina. Centro de Ciências Exatas. Programa de Pós-Graduação em Ciência da Computação. IV. Título.

GABRIEL H. N. E. SILVA

APRENDIZADO DE MÁQUINA ADVERSÁRIO CONTRA DETECTORES DE MALWARE

Trabalho de Conclusão de Curso apresentado ao Curso de Bacharelado em Ciência da Computação do Departamento de Computação da Universidade Estadual de Londrina, como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

BANCA EXAMINADORA

Orientador: Prof. Dr. Bruno Bogaz Zarpelão
Universidade Estadual de Londrina

Prof. Dr. Gilberto Fernandes Junior
Universidade Estadual de Londrina

Gabriel Esteves Messas
Universidade Estadual de Londrina

Londrina, 24 de novembro de 2023.

Este trabalho é dedicado às crianças adultas
que, quando pequenas, sonharam em se
tornar cientistas.

AGRADECIMENTOS

Gostaria, primeiramente, de expressar minha gratidão a Deus por me conceder a força necessária para alcançar esta conquista. Agradeço aos meus pais, Paulo e Rosangela, que sempre me apoiaram incansavelmente e estiveram dispostos a superar todas as barreiras para que eu pudesse alcançar meus sonhos. Aos meus queridos avós, João e Maria, por me inspirarem a estudar e por me darem valiosos conselhos para superar os altos e baixos deste trajeto. Agradeço a minha avó Lourdes por estar presente durante esses anos. Ao meu irmão Matheus, minha irmã Thalita e meu cunhado João Guilherme, muito obrigado por serem partes fundamentais desta jornada e por me darem o suporte que eu precisava. Agradeço a todos os meus colegas de sala, que me auxiliaram em vários momentos durante o curso. Agradeço especialmente aos meus colegas, João Gabriel, Lucca e Vitor Hugo, que estiveram sempre presentes e foram uma fonte constante de apoio e incentivo. Espero ter a honra de tê-los como amigos para a vida toda. Quero também agradecer minha amiga Larissa, obrigado pelos conselhos e o companheirismo. Há cinco anos tínhamos um sonho e conseguimos alcançá-lo. Aos demais amigos e familiares que fizeram parte durante esses anos da minha vida, muito obrigado.

Minha gratidão aos professores do Departamento de Computação pelo seu incansável trabalho e dedicação durante esses anos. Em particular, gostaria de agradecer ao meu orientador, Bruno Bogaz, cujo apoio, orientação e paciência foram fundamentais para a conclusão deste trabalho.

"Não vos amoldeis às estruturas deste mundo, mas transformai-vos pela renovação da mente, a fim de distinguir qual é a vontade de Deus: o que é bom, o que Lhe é agradável, o que é perfeito.
(Bíblia Sagrada, Romanos 12, 2))

SILVA, G. H. N. E.. **Aprendizado de máquina adversário contra Detectores de Malware**. 2023. 53f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) – Universidade Estadual de Londrina, Londrina, 2023.

RESUMO

Os detectores de malware são a principal linha de defesa contra a crescente ameaça dos programas maliciosos. Os programas maliciosos, conhecidos como malwares (*Malicious Software*), possuem o objetivo de roubar informações confidenciais, explorar brechas de segurança e trazer transtornos a usuários. Para auxiliar na detecção desses programas, algoritmos de aprendizado de máquina (*Machine Learning* - ML) podem ser utilizados. O ML, com a combinação de técnicas de análise e extração de dados, possui a capacidade de detectar padrões atribuídos ao comportamento dos malwares. Entretanto, esses algoritmos são suscetíveis a ataques por meio do aprendizado de máquina adversário (Adversarial Machine Learning - AML). Esses ataques possuem a capacidade de evadir ou manipular as classificações dos modelos de ML, possibilitando que programas maliciosos sejam classificados como benignos. Este trabalho tem como objetivo investigar 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, além de entender como a defesa baseada no treinamento pode ser usada combatê-los. Para isso, criaremos um cenário de ataque e outro de defesa que irão nos ajudar a entender o impacto dos ataques aos modelo de ML. Com os resultados obtidos esperamos contribuir com a construção de modelos mais robustos ao AML.

Palavras-chave: Aprendizado Máquina Adversário, Aprendiza de máquina, Detectores de Malware, Malware

SILVA, G. H. N. E.. **Adversarial Machine Learning Against Malware Detectors**. 2023. 53p. Final Project (Bachelor of Science in Computer Science) – State University of Londrina, Londrina, 2023.

ABSTRACT

Malware detectors are the main line of defense against the growing threat of malicious software. Malicious software, known as malware, aim to steal confidential information, exploit security vulnerabilities and bring inconvenience to users. To help detect these programs, machine learning (ML) algorithms can be applied. ML with the help of data analysis and data extraction techniques can detect patterns attached to malware behavior. However, these algorithms are vulnerable through attacks of adversarial machine learning (AML). These attacks have the ability to evade or manipulate the classifications of ML models, allowing malicious programs to be classified as benign. This work has the purpose to study 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 against malware detectors and the defense techniques used to tackle them. For this purpose, we will create an attack scenario and a defense scenario that will help us understand the impact of attacks on the ML model. With the results, we hope to contribute to the construction of models that are more robust to adversarial machine learning.

Keywords: Machine Learning, Adversarial Machine Learning, Malware Scanner, Malware

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AM	Aprendizado de máquina
AML	Aprendizado de máquina adversário
App	Aplicativo
BCA	*Bit Coordinate Ascent*
BGA	*Bit Gradiente Ascent*
FGSM	*Fast Gradient Sign Method*
dFGSM	*Deterministic Fast Gradient Sign Method*
rFGSM	*Randomic Fast Gradient Sign Method*
ML	*Machine Learning*
PE	*Portable Executable*
RF	*Random Forest*
RNN	*Recurrent Neural Network*
SVM	*Support Vector Machine*

SUMÁRIO

1 INTRODUÇÃO

Os computadores como smartphones, computadores pessoais, dispositivos inteligentes, servidores e muitos outros são importantes ferramentas de comunicação e de operação de serviços, e possuem um papel fundamental em diversas áreas da sociedade. Diante dessa importante característica, esses tipos de dispositivos se tornam alvos de diversos ataques cibernéticos que buscam tirar vantagens desses sistemas, aproveitando de falhas de seguranças e da utilização de programas maliciosos, chamados de malwares (*malicious software*), que podem atuar como atores primários nesses ataques. São programas que possuem como objetivo trazer transtornos a usuários, explorar falhas e promover ataques massivos, como no caso das botnets. Esses programas escondem sua verdadeira intenção através de diversas técnicas, enganando usuários e até sistemas de segurança [3] [4] [5].

Para combater a propagação desses programas existem os detectores de malware. Os detectores utilizam algumas técnicas que auxiliam na tentativa de determinar se um programa é malicioso ou não. Uma vez que os malwares conseguem se passar por programas benignos, essa tarefa se torna muito difícil, provocando, muitas das vezes, falsos negativos [6] [3] [4]. Como um reforço aos detectores, no decorrer dos anos passaram a ser utilizadas técnicas de aprendizado de máquina (conhecidas como *Machine Learning-based*) que permitem a análise de padrões encontrados nos mais diversos malwares. Com as análises estática e dinâmica dos programas, e a combinação de técnicas de extração de dados, é possível construir diferentes bancos de dados que podem fornecer informações sobre os programas e suas características. A partir disso, os modelos de aprendizado se tornam capazes de auxiliar na classificação de determinado programa baseado nos padrões que ele apresenta.

Entretanto, ao mesmo tempo em que o aprendizado de máquina (AM) é um reforço aos detectores, ele também é suscetível a ataques por meio de amostras adversárias. As amostras adversárias são bastante conhecidas no universo do AM, permitindo que criadores de malware explorem esta brecha a fim de invalidar sistemas de detecção que utilizam AM. A função dos ataques é subverter e enganar os modelos de AM realizando alterações mínimas em atributos dos programas de forma que um malware seja classificado como benigno [7], mantendo suas propriedades intactas e permitindo que o programa ainda seja nocivo ao sistema alvo [8] [2].

A detecção de malware é um desafio visto que novos malwares ainda podem subverter modelos robustos de detecção como os baseados em aprendizado de máquina. Diante disso, esse trabalho investiga diferentes ataques contra um conjunto de detectores de malware destinados a aplicativos móveis. O experimento é dividido em um cenário alvo,

composto por onze classificadores, dez redes neurais com arquiteturas diferentes e uma Random Forest. Adicionalmente, cinco dessas redes neurais são retreinadas utilizando o treinamento adversário. Também é apresentado um cenário de ataque, no qual são geradas amostras adversárias utilizando técnicas como 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. As redes neurais com treinamento adversário e a Random Forest se mostraram robustas contra esses ataques. Além disso, observamos que a variação na arquitetura dos modelos das redes neurais não teve um impacto significativo na geração de ataques ou na robustez dos modelos.

O restante do trabalho é organizado da seguinte forma: o Capítulo 2 traz a fundamentação teórica de detectores de malware, aprendizado de máquina, aprendizado de máquina adversário e os trabalhos relacionados ao AML na detecção de malwares; o Capítulo 3 apresenta detalhes sobre os experimento, as técnicas de geração de amostra adversárias utilizadas, o treinamento adversário e as métricas utilizadas para avaliar os desempenhos dos modelos; Já o Capítulo 4 apresenta os detalhes dos resultados obtidos no experimento; Por último, o Capítulo 5 encerra o nosso trabalho com as considerações finais.

2 FUNDAMENTAÇÃO TEÓRICA E ESTADO DA ARTE

2.1 Detectores de Malware

Mediante a ameaça dos malwares, os detectores possuem um papel fundamental na linha de defesa contra a propagação e na identificação desse tipo de software. Entretanto, os detectores não são capazes de detectar todos os tipos de malwares. Malwares são alterados facilmente e apresentam características muito semelhantes a programas benignos. Esse problema torna o processo de decidir se determinado programa é malicioso ou não mais complexo [3]. Em suma, os detectores de malware podem ser definidos como sistemas que tentam identificar se determinado programa é um malware ou não [4].

Dadas essas circunstâncias, os detectores de malware se dividem em diversas técnicas que ajudam a analisar os programas. Para entender o funcionamento desses detectores, podemos generalizar o processo de detecção em 3 (três) etapas:

1. *Análise de arquivos*: a análise do malware pode ser feita de maneira estática, dinâmica ou uma combinação de ambas. A análise estática analisa informações dos programas sem a necessidade de executá-los, coletando informações como sequência de bytes, chamadas a bibliotecas e códigos de operação (*opcode*). A análise dinâmica é feita a partir da execução do código em um ambiente seguro. Dessa forma, é possível coletar informações sobre o comportamento do programa enquanto ele está em execução [3] [9].

2. *Extração de atributos*: o processo de extração de atributos utiliza técnicas de mineração de dados para obter informações atreladas a diversos atributos capturados durante as análises [3] [5]. É nessa etapa que é realizada a seleção de atributos. A seleção tem por objetivo remover informações redundantes e irrelevantes, reduzindo a quantidade de atributos. No final desse processo é esperado um conjunto de dados mais significativo que represente informações que possam ser melhor aproveitadas na classificação [5].

3. *Detecção*: após a coleta e filtragem dos dados, é realizada a classificação do programa analisado. Nessa etapa várias técnicas são utilizadas. A escolha da técnica tem influência direta nos tipos e famílias de malwares que serão detectados [5] [3] [4].

Os malwares também podem ser classificados em vários tipos e famílias, considerando as diferentes estratégias e técnicas. Algumas dessas famílias são os vírus, *worms*,

ramsonwares, *botnets*, *spywares* e muitos outros. Cada família é conhecida por ações específicas e podem, inclusive, atuar em conjunto. Para dificultar a sua detecção, os malwares utilizam de diversas técnicas de ofuscação. Essas técnicas podem tornar os malwares ainda mais complexos, aumentando a probabilidade de atuarem livremente nos sistemas.

Algumas das técnicas utilizadas pelos criadores de malware podem ser observadas a seguir:

- *Criptografia*: uso da criptografia para esconder partes importantes do código;
- *Polimórfica*: o método polimórfico permite que o malware possa assumir múltiplas formas. Cópias de malwares desse tipo nunca são iguais. Uma forma de obter esse comportamento é através de uma criptografia variável. A cada nova cópia, um novo algoritmo de criptografia é utilizado, fazendo com que o malware sempre assuma formas distintas.
- *Código morto*: adiciona códigos que não possuem utilidade para a execução do malware, como, por exemplo, funções que nunca são executadas;
- *Furtivo*: recorre a diferentes técnicas para se proteger contra as análises. De forma simples, faz alterações no sistema para fugir da detecção.

Os detectores também possuem diferentes abordagens para combater as técnicas de ofuscação dos malwares. Os dados e atributos minerados durante as fases de análise e extração diferem baseado no tipo de técnica utilizada. Entre as técnicas encontradas na literatura, temos:

- *Baseadas em Assinatura*: observa a sequência de bytes nos arquivos, instruções, *strings* e listas de bibliotecas. Através disso consegue produzir assinaturas únicas para programas. As assinaturas são posteriormente utilizados para detectar malwares já conhecidos. É uma técnica bastante utilizada por ser rápida, porém falha ao tentar detectar malwares ainda desconhecidos pelo sistema [3] [5].
- *Baseadas em Comportamento*: observa o comportamento dos malwares através das chamadas de API, chamadas de sistemas, comportamento da rede, etc.. Resolve parte do problema do método de assinatura, pois, ao analisar o comportamento do programa, pode identificar variações da mesma família de malwares que adicionam, por exemplo, código morto e alteram sua assinatura. As desvantagens desse método estão ligadas ao tempo de processamento dos programas e à quantidade de programas benignos que são classificados equivocadamente como malwares [6] [3].

- *Baseadas em Heurística*: utiliza uma combinação de técnicas, entre elas o uso de algoritmos de AM, além da aplicação de mineração de dados na extração de dados. Analisa chamadas de API e chamadas de sistemas que não são comumente utilizadas. A partir disso, se baseia em uma série de regras que definem limites ao comportamento dos programas. Quando um limite é extrapolado, aquele programa pode apresentar características de um malware. É um modelo mais robusto e proativo na detecção, sendo bastante aplicado no mercado devido à sua capacidade de lidar com mutações de malwares da mesma família e detectar novos malwares que seguem comportamentos semelhantes. Entretanto, apresenta alta taxa de falsos positivos e falha ao detectar malwares que são bastante complexos, além da criação das regras consumirem bastante tempo [3] [5].

Além dessas técnicas existentes, temos outras como *mobile-based* e *cloud-based*, que cresceram recentemente devido à utilização de dispositivos móveis e a fácil escalabilidade dos ambientes em nuvem. A aplicação do AM para a classificação também se tornou uma ferramenta bastante efetiva atualmente auxiliando no processo de detecção. O sucesso dos detectores baseados em AM dependem fortemente das fases de análise e extração de atributos já que os algoritmos de AM são orientados a dados [10]. A escolha do algoritmo utilizado pelo modelo de AM também influencia no desempenho dos detectores, visto que o desempenho de diferentes algoritmos muda conforme a distribuição dos dados e a quantidade de atributos nos conjuntos de dados analisados [5]. A Figura 1 mostra um esquema simples de um detector baseado em AM. As fases de análise e extração de dados irão fornecer os dados necessários, já rotulados, para o treinamento e teste do modelo. Em seguida, é esperado que o modelo seja capaz de generalizar o que foi aprendido para dados desconhecidos.

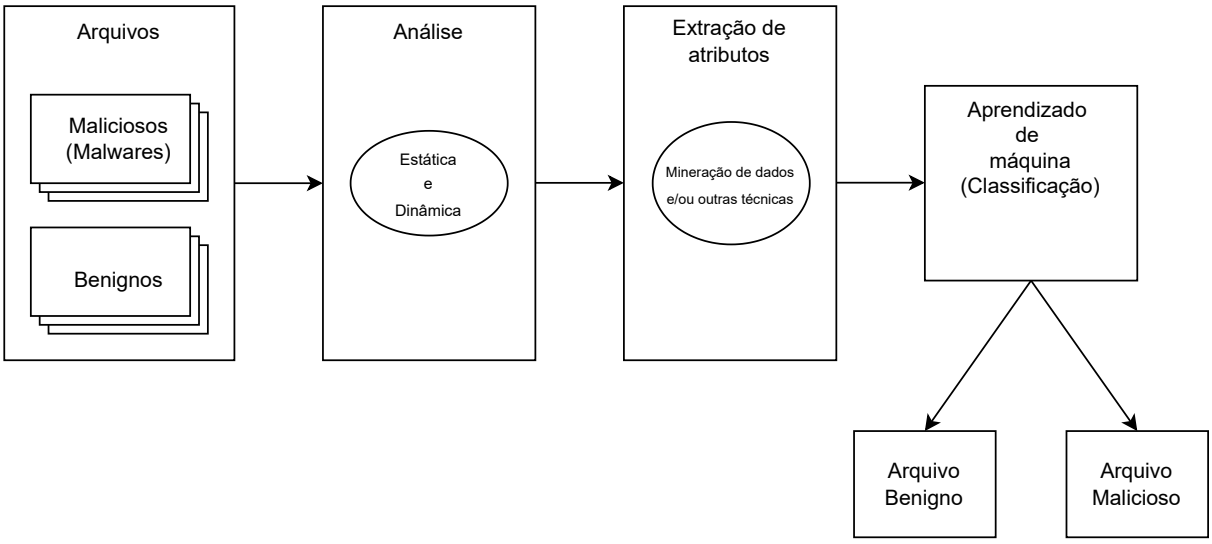

Figura 1 – Exemplo de um detector baseado em Machine Learning.

2.2 Aprendizado de Máquina (AM)

O aprendizado de máquina estuda algoritmos que melhoram com experiência [10]. São excelentes ferramentas de regressão e classificação de dados, visto que se ajustam ao aprendizado obtido sobre o conjunto de dados e não necessitam serem explicitamente programados para isso [11] [12]. Como ferramenta de regressão, os algoritmos tentam prever um valor contínuo, como, por exemplo, salário de um funcionário ou a idade de uma pessoa. Como ferramenta de classificação, os valores são categóricos, por exemplo, dizer se o comportamento de um programa é malicioso ou não. Os algoritmos de AM podem ser divididos em duas categorias: aprendizado supervisionado e aprendizado não supervisionado.

No aprendizado supervisionado, podemos dividir o processo em duas etapas: aprendizado e inferência. No aprendizado, o algoritmo de AM recebe um conjunto de dados rotulado como entrada e realiza operações estatísticas sobre eles. Ao final desse processo, é criado um modelo que tenta predizer o rótulo pertencente a determinada entrada que compartilha similaridade com os dados utilizados nas operações. O objetivo é que os rótulos previsto pelo modelo se aproximem dos rótulos reais [12]. Para atingir esse objetivo, o modelo vai sendo ajustado durante o aprendizado para que os rótulos previstos coincidam cada vez mais com os rótulos verdadeiros. Após o aprendizado, é possível realizar inferências sobre dados que não foram utilizados na fase de aprendizado. Nessa fase são coletadas métricas sobre o desempenho do modelo durante a classificação. Como exemplos de algoritmos de aprendizado de máquina temos as redes neurais, árvores de decisão e as máquinas de vetor de suporte.

O aprendizado não supervisionado é aplicado em situações onde é custoso rotular os dados. Por exemplo, em um fluxo de rede onde existe um grande tráfego de dados. Nesse caso, os dados que apresentam características subjacentes semelhantes são colocados em grupos. Esses grupos são chamados de *clusters*. Esse processo conhecido como clusterização fornece informações sobre a organização e comportamento dos dados [12].

2.2.1 Redes Neurais

As redes neurais são um subconjunto do aprendizado de máquina. Possuem esse nome por conta da sua estrutura ser baseada no funcionamento do cérebro humano, com neurônios que se conectam e conversam entre si por meio das sinapses [13]. Uma rede neural pode ser organizada da seguinte maneira: uma camada de entrada, camadas ocultas e uma camada de saída. A Figura 2 exemplifica a organização da rede.

A camada de entrada é um conjunto de neurônios onde cada neurônio recebe

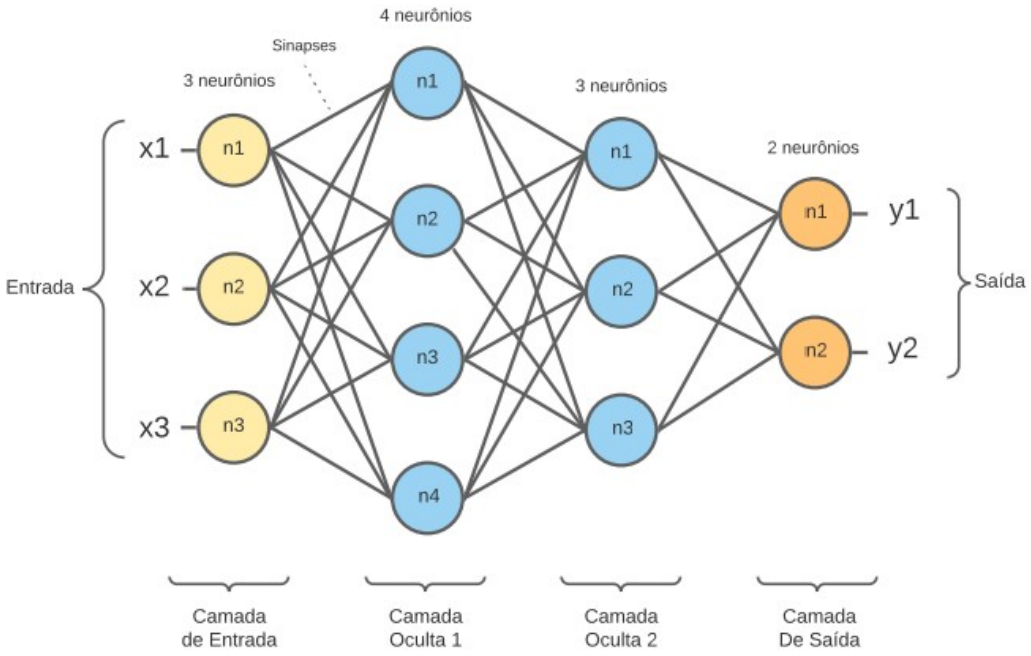

Figura 2 – Exemplo ilustrativo de uma rede neural com 3 entradas, 2 camadas ocultas e 2 saídas.

um atributo do conjunto de dados. As saídas desses neurônios se conectam às entradas dos neurônios da primeira camada oculta. Na camada oculta é aplicada a função de ativação, que está presente dentro de cada neurônio e decide quando ele deve ser ativado ou não [14]. Cada neurônio possui um peso e um viés (ou limite) atrelado a ele. O peso é multiplicado pelo valor de entrada. A função de ativação verifica se o valor obtido ultrapassa o limite estipulado e, caso ultrapasse, o neurônio é ativado. Quando ativado, o neurônio transmite informações para o próximo neurônio com quem ele se conecta. Uma rede neural pode possuir quantas camadas ocultas e neurônios por camada forem desejados. Normalmente, quando existem mais de 3 camadas ocultas, as redes neurais são chamadas de *Deep Neural Network* (Rede Neural Profunda) [13]. Os neurônios da última camada oculta se conectam com os neurônios da camada de saída, que produz o resultado das operações. A quantidade de neurônios na saída é dada pela quantidade de classes no problema analisado. O processo de transmissão de informações de um neurônio para o outro na próxima camada é conhecido como *feedfoward*.

As redes neurais aprendem realizando ajustes nos pesos e vieses dos neurônios. Para isso, ao final da computação de um conjunto de amostras de treinamento é realizada uma comparação entre os valores estipulados com o valores reais das saídas. A comparação é feita por uma função de custo que irá dizer o quão errado o modelo cria relações entre os valores de entrada e os valores de saída. O resultado da função de custo é repassado para os neurônios reajustarem seus pesos e vieses em um processo chamado de *backpropagation*. O objetivo é que o erro da relação entre a entrada e a saída seja reduzido [12].

2.2.2 Random Forest

O Random Forest é um algoritmo de aprendizado de máquina baseado na combinação de várias árvores de decisão. Cada árvore é construída de maneira independente utilizando um subconjunto aleatório de atributos. Os atributos são obtidos de amostras, também aleatórias, igualmente distribuídas do conjunto de dados. A ideia geral é que esse conjunto de árvores possa mitigar o erro individual gerado por cada árvore e convergir para um modelo mais robusto [15]. A técnica de combinar diversos classificadores em um modelo mais robusto é conhecida como *ensemble*.

Podemos entender uma árvore de decisão sendo formada por nós de decisão e folhas. Nós de decisão realizam testes que definem um caminho a ser tomado. A direção do caminho escolhido leva a outros nós de decisão até que chegue a uma folha, ou seja, a uma decisão. Cada nó de decisão representa um atributo selecionado. A folha representa um estado final pertencente ao conjunto de classes do problema. A Figura 3 é um exemplo de uma árvore de decisão simples.

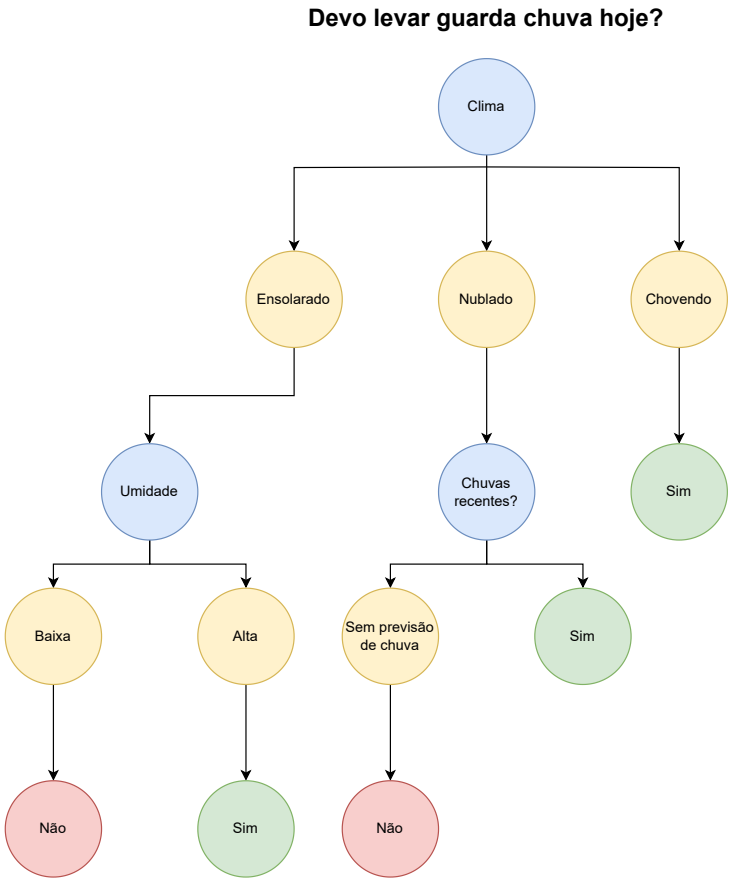

Figura 3 – Árvore de decisão para determinar a utilização de um guarda chuva

É necessário que seja estabelecido um critério de divisão dos nós durante a criação da árvore. Este critério é utilizado para selecionar a melhor forma de dividir os atributos, a fim de garantir que cada sub-ramo agrupe características semelhantes, mas sejam distintos

entre si [16]. Entre os tipos de critérios, temos:

- Gini: Este método calcula a probabilidade de uma amostra aleatória ser corretamente rotulada por uma classe i em um nó denotado por p_i. Essa função pode ser interpretada como o grau de impureza desse nó, conhecido como *GiniImpurity*. Essa relação pode ser entendida na equação 2.1.

$$GiniImpurity = 1 - \sum_{i=1}^{k} p_i^2 \tag{2.1}$$

  Nesse caso, quanto menor for a impureza do nó, mais próximo de um grupo homogêneo ele se encontra. Entende-se como grupo homogêneo, ou puro, um grupo que é composto apenas por atributos que convergem para a mesma classe.

- Entropia ou Ganho de informação: O método de entropia é baseado no ganho de informação e, semelhante com o Gini, calcula a pureza, ou, o quão homogêneo é um nó. A Equação 2.2 demostra como podemos obter a entropia de uma árvore. Quanto menor for a entropia, mais homogêneo é um grupo.

$$Entropy = -\sum_{i=1}^{k} p_i \log_2(p_i) \tag{2.2}$$

O Random Forest pode ser aplicado tanto para regressão quanto para a classificação. Em ambos os casos as árvores são construídas com técnicas semelhantes. Entretanto, na regressão as árvores estipulam um valor contínuo. Dessa forma, podemos estimar o valor final como uma média das respostas de cada árvore. Na classificação, os valores estipulados pelas árvores são categóricos, ou seja, a qual classe determinada entrada pertence. Cada árvore faz sua estimativa e, no final, a classe com a maior frequência é escolhida. A ideia para ambos os casos é diminuir o erro das saídas, ou seja, encontrar uma combinação de parâmetros ótima para o modelo. Os parâmetros buscam definir, por exemplo, a quantidade de árvores no modelo, a quantidade de atributos por árvores, a profundidade delas e seu critério de divisão [17]

2.2.3 Support Vector Machine (SVM)

Support Vector Machine (SVM) é um algoritmo de aprendizado de máquina amplamente utilizado para problemas de classificação e regressão. Ele se baseia na ideia de encontrar um hiperplano ótimo para separar os dados em duas classes, maximizando a distância entre elas. Como ilustrado na Figura 4, a tarefa de encontrar o hiperplano ideal pode parecer simples, pois existem inúmeras opções disponíveis. No entanto, é importante encontrar o hiperplano que melhor se adapte aos dados para garantir uma boa generalização dos resultados [18].

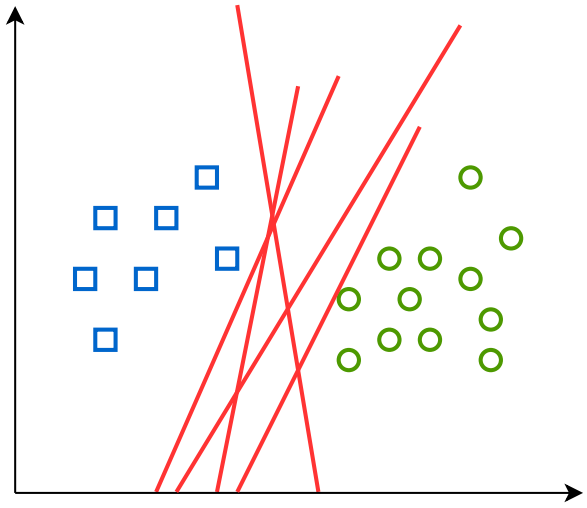

Figura 4 – Diferentes linhas (hiperplanos) que separam os dados

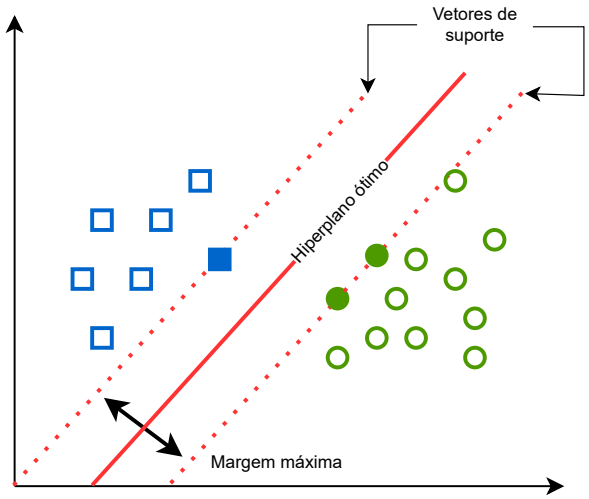

Figura 5 – Hiperplano ideal utilizando o SVM

Os pontos presentes nas fronteiras entre as classes, representados na Figura 5, são chamados de vetores de suporte. Esses pontos são fundamentais para estabelecer a margem ideal para o hiperplano. O algoritmo da SVM precisa encontrar um equilíbrio entre a margem ideal e o erro do modelo, o que pode resultar em classificações incorretas em alguns casos, para maximizar a distância entre as classes. Esse processo é conhecido como suavização de borda. A margem atua como uma medida de confiança na classificação, quanto maior a distância de um ponto à margem, maior a confiança na classificação do modelo [19] [20].

2.3 Aprendizado de máquina adversário

O Aprendizado de máquina adversário (*Adversarial machine learning* - AML) é uma área que estuda como os algoritmos de aprendizado de máquina podem ser atacados e defendidos. A intenção é criar amostras adversárias, que são dados especificamente manipulados, para enganar e confundir os modelos. Essas amostras são elaboradas por diversas técnicas de ataques. Através do estudo e entendimento desses ataques, é possível explorar as fraquezas que um modelo pode apresentar e assim novas defesas podem ser construídas. O objetivo do AML é aumentar a robustez dos modelos, tornando-os mais resistentes aos ataques [21].

2.3.1 Tipos de ataques

Os ataques baseados no AML possuem diferentes abordagens e podem atuar com o conhecimento de informações do modelo-alvo ou não. Quando o atacante possui conhecimento dos parâmetros do modelo e do processo de treinamento, esse ataque é chamado *white box*. Caso contrário, é chamado de *black box*. Os ataques *black box* são os mais comuns em cenários reais, uma vez que os atacantes frequentemente não possuem fácil acesso às informações do modelo. Apesar disso, segundo o conceito de *transferability*, atacantes podem aprender a manipular de maneira eficiente as entradas utilizando um algoritmo de AM diferente daquele utilizado pela vítima. Em outras palavras, o conhecimento obtido pelo atacante com um algoritmo pode ser transferido para atacar outros algoritmos. Dessa maneira, apesar da falta de informação, ataques *black box* também fornecem um meio viável e possível para atacar modelos de AM [22] [23].

Podemos categorizar os ataques por finalidades diferentes. Eles podem buscar evadir classificações, como no caso dos ataques de evasão, ou influenciar no desempenho dos modelos, como é o caso do ataque de envenenamento. O ataque de envenenamento (*poisoning attack*) tem como objetivo manipular o conjunto de treinamento de um modelo de aprendizado de máquina, inserindo dados maliciosos ou manipulados, de modo que o modelo seja treinado incorretamente ou tenha sua precisão reduzida. Já o ataque de evasão (*evasion attack*) consiste em enganar o modelo de aprendizado de máquina no momento em que ele está em execução, apresentando dados de entrada especialmente criados para que o modelo produza uma saída errada ou não detecte uma ameaça. Em resumo, o ataque de envenenamento ocorre na fase de treinamento, enquanto o ataque de evasão ocorre na fase de inferência ou aplicação do modelo. A Figura 6 categoriza essas informações.

Os ataques de envenenamento podem ser indiscriminados se o objetivo é afetar qualquer classe, ou de integridade, se o objetivo for afetar determinadas classes. Quando o ataque é de integridade, existe a possibilidade da criação de *backdoors* nos modelos-alvo. Um *backdoor* é basicamente uma falha de segurança deixada pelo atacante no sistema alvo o possibilitando de realizar outros ataques posteriormente. Na detecção de malware,

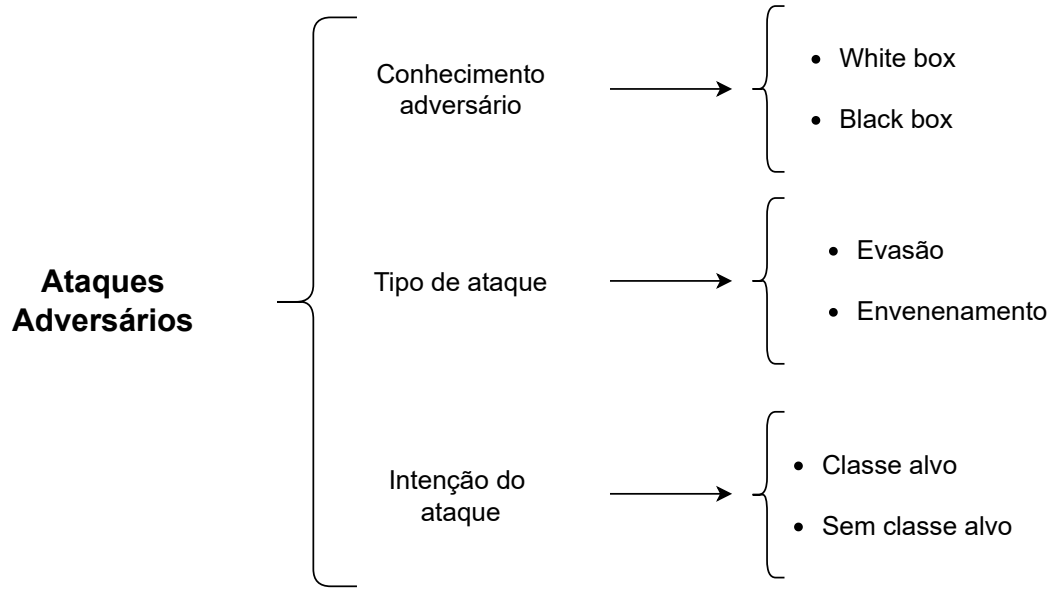

Figura 6 – Categorização dos ataques adversários. Imagem adaptada de Shaukat *et al.* [1]

por exemplo, foi demonstrado por Yang *et al.* [23] um ataque que permite um *backdoor* durante a classificação de arquivos. Este ataque permitiu que arquivos maliciosos aproveitassem uma vulnerabilidade no sistema, fazendo com que eles fossem classificados como inofensivos enquanto outros malwares continuavam a ser corretamente identificados como maliciosos.

Em ambos os casos, podemos melhor definir uma amostra adversária como na equação 2.3, onde x é uma amostra benigna perturbada por um ruído δ a fim de gerar uma amostra adversária x^* [7]. A maneira de se obter δ irá depender da estratégia abordada pelo ataque.

$$x^* = x + \delta \tag{2.3}$$

Por exemplo, uma das maneiras de encontrar δ é aplicando o ataque FGSM (*Fast Gradient Sign Method*). Esse ataque se baseia na função de otimização dos modelos. O FGSM utiliza o gradiente da função de erro do modelo para determinar a direção na qual a entrada deve ser perturbada para causar a maior mudança na saída do modelo [22]. Quando falamos em função de erro, ela está relacionada a otimização da função de custo utilizada pelos algoritmos. Através disso, no caso do FGSM, o atacante determina um valor ϵ, suficiente, que será utilizado para perturbar a entrada. Em um primeiro momento, esse tipo de ataque gera ruídos que não fazem sentido em um domínio discreto como é o caso dos malwares, entretanto, uma transição de domínio pode ser realizada e, nesse caso, determina-se quais atributos serão manipulados ao modelo para afetar sua capacidade preditiva [1].

Outra maneira de gerar amostras adversárias é utilizando a matriz Jacobiana. Os ataques baseados na matriz Jacobiana são amplamente conhecidos na classificação de imagens e foram adaptados para a classificação de malwares por Grosse *et al.* em [24] [25]. Essa técnica mede a sensibilidade da saída de uma função em relação às suas entradas. A função no caso é a função de erro do modelo. O objetivo é maximizar o erro dessa função selecionando apenas alguns atributos presentes no dado de entrada. Esse ataque é semelhante ao FGSM, mas leva em consideração uma estratégia diferente para efetuar a perturbação nos dados.

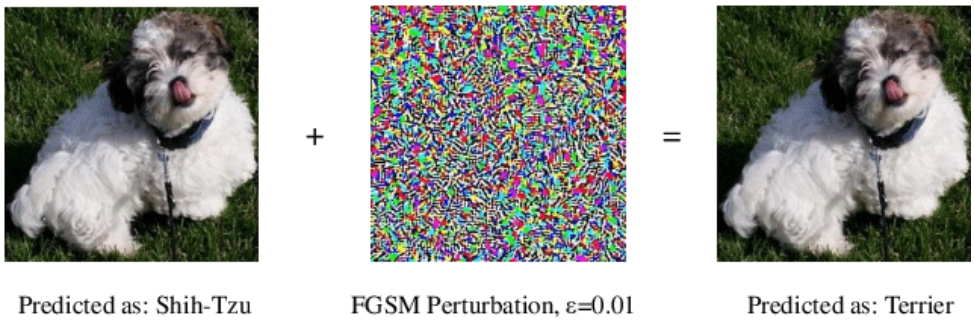

Figura 7 – Ataque FGSM (baseado em gradiente) aplicado na visão computacional. A figura mais à esquerda representa o dado de entrada sem perturbação sendo classificado como um Shih-Tzu. A figura central mostra o ruído obtido utilizando o FGSM com um $\epsilon = 0.01$. Essa perturbação é adicionada a todos os pixeis da imagem. Por fim, a última imagem nos apresenta a imagem perturbada sendo classificada como um Terrier.

Quando nos concentramos nos ataques contra detectores de malware, surgem novos desafios ao atacante. As pequenas modificações que são geradas pelos ataques precisam garantir que o malware mantenha suas funcionalidades essenciais. Com as manipulações nos atributos, os arquivos podem perder sua funcionalidade maliciosa, tornando o ataque ineficiente [24] [26] [27]. Essa área de aplicação também apresenta um problema adicional. Os ataques comumente demonstrados na visão computacional [22] diferem da estrutura dos arquivos binários de programas de computador, utilizados em modelos de classificação de malwares. Enquanto as imagens são compostas por matrizes de pixels, os programas possuem uma organização específica, que varia de acordo com o sistema de origem. Assim, para construir ataques contra sistemas de detecção de malwares, é necessário analisar a organização dos dados a fim de preservar a semântica do programa [26, 8].

De modo geral, quando utilizando arquivos binários que compõem o software a ser analisado, o treinamento é realizado sobre um vetor de atributos. Um vetor de atributos pode ser entendido também como um vetor binário. Ele mapeia todos os atributos presentes no conjunto de dados e, para determinado arquivo, quando um atributo está presente, um dígito inteiro 1 é atribuído a posição do atributo no vetor, caso contrário, 0. A Figura 8 nos dá um exemplo de como é formado um vetor de atributos. Portanto, as perturbações realizadas sobre o vetor de atributos binários devem ser representadas por

valores binários, isto é, 0 ou 1, indicando a ausência ou presença, respectivamente, de um determinado atributo.

Atributo 1	Atributo 2	...	Atributo n-1	Atributo n
0	1	...	0	0

Figura 8 – Exemplo de um vetor de atributos. O valor 1 representa a presença de um atributo no arquivo, enquanto o valor 0 indica sua ausência.

2.3.2 Técnicas de defesa

As técnicas de defesa têm por objetivo aumentar a robustez dos classificadores, assim tornando-os mais resistentes aos ataques. As defesas encontradas na literatura são normalmente aplicadas na área de visão computacional, mas também podem ser transpostas aos detectores de malware como apresentado por [2]. As técnicas de defesa atuam principalmente na fase de treinamento dos classificadores como, por exemplo, o treinamento adversário, sugerido por [22] e o *defensive distillation* [28].

O treinamento adversário determina que amostras adversárias sejam inseridas durante a fase de treinamento. Assim, os classificadores aprendem a classificar possíveis padrões relacionados a certos tipos de ataques. Entretanto, como apontado por [2] e [27], as amostras precisam ser cuidadosamente selecionadas, pois os algoritmos podem aprender a classificar determinado padrão de ataque e passar uma falsa sensação de proteção.

O *defensive distillation* utiliza um modelo a mais durante a fase de treinamento. Essa técnica, de forma simples, normaliza as saídas obtidas pelo primeiro modelo e utiliza esses valores como rótulos para os dados de entrada. Em seguida, esse conjunto é utilizado para treinar o próximo modelo que recebe o nome de modelo destilado [24]. Segundo [28] a experiência obtida não é apenas concentrada nos parâmetros de pesos da rede neural, mas também pelo vetor de probabilidade[1] produzido pelo modelo.

2.4 Trabalhos relacionados

Os trabalhos que estudam e investigam as técnicas de aprendizado de máquina adversário tiveram início no campo da visão computacional, na classificação de imagens [29] [22] [25]. Em pesquisas posteriores, os estudos expandiram para outras áreas de interesse, como no caso da detecção de malwares. Em todos os diferentes ambientes, novos desafios são enfrentados. No caso dos detectores de malwares, os estudos lidam com um cenário em constante desenvolvimento, como mencionado na seção 2.1. Os trabalhos selecionados para compor a base teórica desse estudo apresentam diversas metodologias e técnicas que

[1] O vetor de probabilidade é criado por uma função de ativação na última camada de saída da rede neural. Essa função, conhecida como softmax, estipula a probabilidade da entrada pertencer as k classes do problema.

visam analisar e compreender o comportamento do aprendizado de máquina adversário em relação aos detectores de malwares.

Grosse *et al.* [2] [24] propõem um novo ataque baseado no cálculo da matriz jacobiana. Este ataque desloca a ideia já concebida anteriormente na visão computacional para o domínio da cibersegurança, em específico, a detecção de malwares. O modelo alvo é uma rede neural profunda que alcançou bons resultados na classificação de malwares. O ataque proposto leva em consideração apenas a classificação dos malwares pelo modelo alvo, sendo assim considerado um ataque *black box*. O estudo é baseado em um conjunto de dados de aplicativos Android conhecido como Drebin. Além disso, o foco do ataque é apenas em adicionar novos atributos estáticos às entradas perturbadas.

Rosenberg *et al.* [30] propõe um ataque genérico e *black box* contra classificadores de malwares considerados estado da arte. Este ataque utiliza o conceito de *transferability* para criar um modelo substituto e tem como alvo alguns algoritmos como o RNN (Rede Neural Recorrente), DNN, SVM e Random Forest. Diferente de Grosse *et al.*, Rosenberg *et al.* gera amostras adversárias para atributos estáticos e dinâmicos. Seu ataque opera com a adição de chamadas API que não possuem efeito nas funcionalidades do malware. A seleção das APIs a serem adicionadas é realizada com base no impacto gerado na predição do modelo. Essa informação é estipulada utilizando uma matriz Jacobiana que é extraída do modelo substituto. O modelos substituto, é o modelo utilizado pelo atacante para simular o modelo alvo. Isso é possível por causa do conceito de *transferability* mencionado na Seção 2.3.1. Para a realização do estudo foi utilizado um conjunto de dados de arquivos PE (*Portable Executable*), um padrão de arquivos para o sistema operacional *Windows*.

Pierazzi et al. [26] também realiza um estudo utilizando arquivos Android. Seu trabalho propõe uma nova ideia para aumentar a eficácia dos ataques contra detectores de malwares. Ataques como o proposto por Grosse *et al.*, Rosenbeg *et al.* e Shaukat *et al.* podem ser anulados por defesas externas como o pré-processamento de dados e remoção de APIs ociosas. Pierazzi contorna isso propondo uma técnica de camuflagem de malwares. Esta técnica consiste em inserir partes de códigos benignos sem afetar a funcionalidade maliciosa dos malwares, a fim de deixá-lo com uma aparência e comportamento inofensivo. Este método se mostrou efetivo contra o Sec-SVM, uma versão mais robusta do SVM, considerada estado da arte na detecção de malwares.

Al-Dujaili *et al.* [31] propõem dois ataques adversários, o BGA (Bit Gradient Ascent) e o BCA (Bit Coordinate Ascent), para o domínio binário, e comparam sua eficácia com métodos já existentes na literatura, como dFGSM, rFGSM e o método de Grosse. O experimento é realizado em arquivos no formato PE, comparando a eficácia dos ataques em modelos sem defesa e em modelos defendidos com treinamento adversário.

O trabalho desenvolvido por Shaukat et al. [1] propõe um método para aumentar a robustez detectores de malware por meio do treinamento adversário. O seu método

combina 10 tipos de ataques diferentes, sendo eles o FGSM e algumas variações como rFGSM e dFGSM, BGA, BCA, DeepFool, método de Grosse, BIM, o PGD e um ataque combinado com várias técnicas para gerar as amostras. Muitos desses ataques são observados na classificação de imagens como é o caso do DeepFool. Shaukat *et al.* realizam uma série de experimentos observando o impacto dos ataques contra diferentes treinamentos de uma rede neural. Seu modelo alvo é treinado várias vezes com amostras de cada um dos ataques, sendo considerados 10 tipos de modelos. No trabalho foram utilizados dois conjuntos de dados diferentes para arquivos no formato PE. O ataque combinado foi bastante efetivo na maioria das vezes, tendo sua performance reduzida quando atacando os modelos treinados com amostras dos ataques rFGSM, Grosse e BCA.

No que tange as técnicas de defesa, os trabalhos de Grosse *et al.* e Shaukat *et al.* investigam principalmente o treinamento adversário e apresentam bons resultados que contribuíram para uma maior robustez dos modelos. No experimento de Shaukat *et al.*, os modelos treinado com as amostras adversárias dos ataques combinados, rFGSM, Grosse e BCA apresentaram um diminuição na taxa de evasão das amostras aumentando a robustez dos modelos. Grosse *et al.* se aprofundam ainda mais e analisam o *defensive distillation*. Segundo seu estudo, o *distillation* foi efetivo contra as amostras adversárias, mas não o suficiente para ser considerado uma técnica segura. Pierazzi *et al.* demonstram que as técnicas de defesa podem ir além, com as fases que antecedem a classificação de um malware podendo ser cruciais para a eliminação do malware. Seu ataque se mostrou resistente a essa estratégia e eficiente contra o modelo alvo. Apesar disso, aumentar a segurança do modelo ainda é uma tarefa a ser explorada.

Os trabalhos citados até este ponto são ataques de evasão, ou seja, ataques realizados durante a fase de inferência. O trabalho de Yang *et al.* [23] desenvolve um ataque de envenenamento na detecção de malwares. A intenção do ataque é criar um *backdoor* seletivo nos modelos que pode ser explorado pelo atacante posteriormente. O modelo alvo é um *gradient boosting*, uma técnica de aprendizado de máquina baseado em árvore de decisão. Os resultados demonstraram eficácia contra o modelo que foi reforçado com defesas específicas de detecção de *backdoors*.

Além dos trabalhos mencionados anteriormente, o estudo de Pendlebury *et al.* [32] apresenta considerações fundamentais sobre o viés presente nos experimentos de classificação de malwares. O objetivo principal desse estudo é evidenciar a diferença entre o cenário real e o apontado em outros artigos. Para isso, eles propõem uma abordagem baseada em técnicas de distribuição espacial e temporal dos conjuntos de dados, visando aumentar a eficácia dos sistemas de detecção de malwares em ambientes reais, ao considerar as necessidades e desafios enfrentados nesses ambientes.

Os trabalhos de Shaukat *et al.*, Al-Dujaili *et al.* e Rosenberg *et al.* investigam classificadores de arquivos PE, que são específicos para o sistema operacional Windows.

O trabalho de Grosse *et al.*, por sua vez, concentra-se em aplicativos Android, mas se restringe apenas ao ataque proposto em seu trabalho. O presente trabalho tem como objetivo utilizar aplicativos de dispositivos móveis baseados no sistema operacional Android, já que esses dispositivos são alvos frequentes de ataques devido à sua popularidade. Assim como Shaukat *et al.* e Al-Dujaili *et al.*, utilizamos os ataques dFGSM, rFGSM, BGA, BCA e Grosse contra redes neurais com e sem treinamento adversário. Entretanto, este trabalho busca comparar o desempenho das redes com um algoritmo mais robusto, como o Random Forest. Além disso, enquanto os estudos anteriores se limitam a apenas uma arquitetura de rede neural, é explorado se variações nas arquiteturas das redes tornam os modelos menos suscetíveis ou não a amostras maliciosas.

3 MATERIAIS E MÉTODOS

Neste capítulo iremos descrever os materiais e métodos utilizados para realizar o estudo proposto neste trabalho. Serão explicadas as escolhas tomadas em relação aos dados utilizados, o experimento proposto, como realizaremos a análise e avaliação dos resultados.

3.1 Conjunto de dados

O conjunto de dados utilizado neste trabalho foi criado e fornecido pelo projeto Drebin[1]. Este projeto tem como objetivo contribuir para a detecção de malwares em dispositivos móveis que utilizam o *Android*, propondo um método de detecção de baixo impacto nos dispositivos, mas com desempenho comparável ao estado da arte. Para a elaboração do conjunto de dados, foram coletados 129.013 aplicativos reais de várias lojas de aplicativos durante o período de agosto de 2010 a outubro de 2012. A coleta de dados abrangeu diversos tipos de aplicativos *Android*, a fim de tornar o conjunto de dados representativo do ambiente real de utilização de dispositivos móveis. Esses aplicativos foram então rotulados como benignos ou maliciosos por meio de diversas técnicas de detecção, incluindo o próprio Drebin. Dessa forma, o conjunto de dados é composto por 123.453 aplicativos benignos e 5.560 maliciosos (malwares). Além disso, os malwares foram identificados pertencendo a 179 tipos de famílias diferentes. Cada família de malware possui suas estratégias e objetivos [33].

Para a extração dos atributos, foi realizada uma análise estática sobre os aplicativos. Essa análise levou em consideração apenas o manifesto e o código dex do aplicativo. O manifesto é responsável por armazenar informações essenciais do app. Nele é possível encontrar os componentes utilizados, as permissões requisitadas e os componentes de hardware. O código dex é um formato de arquivo binário utilizado pelo sistema operacional *Android*. Ele é necessário para que o aplicativo seja executado pelo sistema.

A partir da análise, os atributos extraídos foram organizados em 8 grupos, cada um deles representando um tipo de informação. Os grupos são identificados pela nomenclatura S_x, onde x é o número do grupo, assim temos:

- S_1 Componentes de hardware: os hardwares necessários para o funcionamento do aplicativo;
- S_2 Permissões: permissões que serão utilizadas pelo aplicativo;

[1] Disponível em: https://www.sec.tu-bs.de/ danarp/drebin/index.html

Tabela 1 – Grupos de atributos e o local onde são extraídos. Tabela adaptada de Grosse *et al.* [2]

ID	Nome	Manifesto	Cód. dex
S_1	Componentes de Hardware	X	
S_2	Permissões	X	
S_3	Componentes	X	
S_4	Filtros de Intents	X	
S_5	Chamadas API restritas		X
S_6	Permissões Utilizadas		X
S_7	Chamadas API suspeitas		X
S_8	Endereços de rede		X

- S_3 Componentes: são os blocos de construção de um aplicativo *Android*. Existem 4 tipos de componentes: atividades, serviços, *broadcast receivers* e provedores de conteúdo. Cada tipo define uma interface diferente no sistema. Um aplicativo pode declarar esses componentes várias vezes. Esse grupo armazena os nomes utilizados para identificar esses componentes;
- S_4 Filtros de intents: estrutura de dados que é responsável pela comunicação entre componentes;
- S_5 Chamadas API restritas: algumas permissões do *Android* são listadas como críticas e devem ter autorização do usuário para serem utilizadas;
- S_6 Permissões utilizadas: requisições que foram permitidas pelo usuário e estão sendo utilizadas. Podemos entender este grupo como um subgrupo de S_5;
- S_7 Chamadas API suspeitas: são APIs que acessam dados ou recursos sensíveis como o identificador do dispositivo e acesso a funções de comunicação de rede;
- S_8 Endereços de Rede: Composto de URL, IPs e *hostnames*, são endereços para acessar dados na rede.

Apesar de existir grupos similares, como o S_5 e S_2, as informações presentes em cada um são coletadas de locais diferentes. Como apresentado na Tabela 3.1, os atributos dos grupos S_1 até S_4 foram extraídos do manifesto, enquanto os grupos S_5 até S_8 são extraídos do código dex. Os grupos existem como uma referência para identificar o local de onde os atributos foram extraídos. Os atributos serão posteriormente transformados no vetor de atributos, como exemplificado na Figura 8. Os atributos extraídos de ambos os arquivos podem ser entendidos como características de um aplicativo. Elas descrevem como aquele aplicativo é executado e como ele funciona.

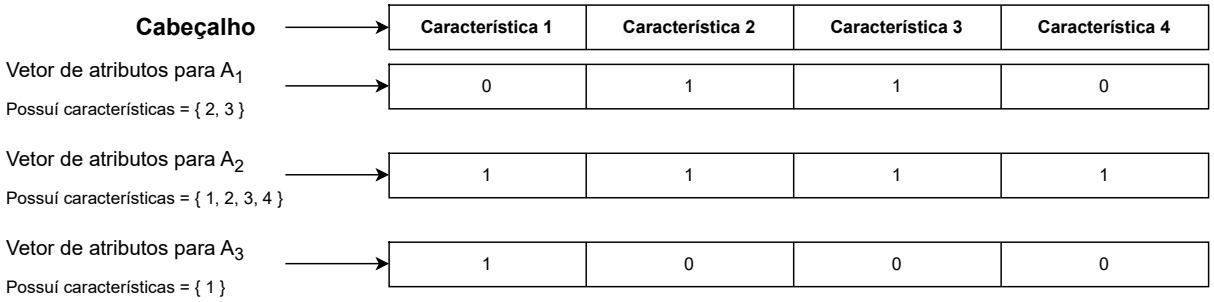

Figura 9 – Vetor de atributos construído a partir das características extraídas de diferentes aplicativos.

Em um momento antes da fase de treinamento é definido um cabeçalho que mapeia essas características para uma posição no vetor de atributos. Esse cabeçalho serve apenas como suporte. Podemos tomar esse cabeçalho como um conjunto C, tal que, $C = A_1 \cup A_2 \cup \cdots \cup A_{n-1} \cup A_n$, onde A_x é o conjunto de características extraídas de um aplicativo x. Para cada aplicativo x será construído um vetor de atributos. Esse vetor atribuirá o valor 1, presença de uma característica, para as posições das características resultantes de $C \cap A_x$ e o valor 0, caso a característica não esteja presente na intersecção. A Figura 9 é um exemplo visual de como isso ocorre.

A distribuição de dados é bastante desequilibrada. Observando a Figura 10, é possível ter uma ideia da quantidade de aplicativos benignos em relação aos malwares. A proporção inicial do conjunto é de aproximadamente 22 benignos para 1 malware, sendo que os benignos equivalem a 95% dos dados. Neste trabalho foram utilizados 120 mil aplicativos benignos e 5 mil malwares, aumentando a proporção para 24:1, ou seja, a cada 24 benignos existe 1 malware. Essa escolha do aumento da proporção foi baseada após a remoção de alguns arquivos corrompidos.

3.2 Experimento

O experimento proposto neste trabalho tem como objetivo investigar e explorar dois cenários diferentes, respondendo a quatro perguntas:

1. Qual a severidade dos ataques?
2. Qual a eficácia do mecanismo de defesa?
3. Qual a influência da arquitetura do modelo no ataque e defesa?
4. Qual o desempenho da Random Forest comparado às redes neurais?

Para responder a essas perguntas, foi feita a divisão do experimento em um cenário de ataque e um cenário alvo. No cenário de ataque, são geradas amostras adversárias

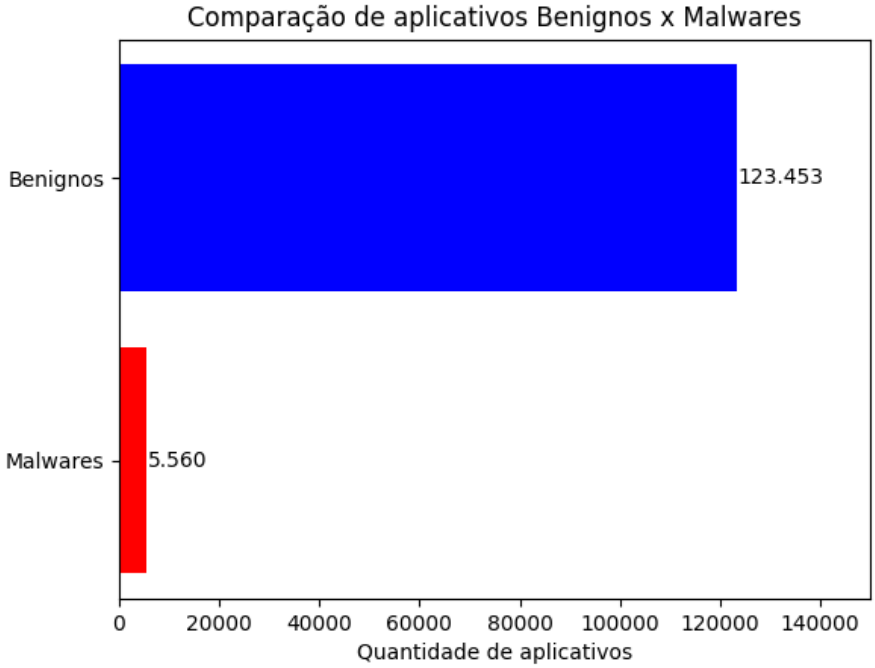

Figura 10 – Comparação da quantidade de benignos e malwares no conjunto de dados.

que serão classificadas pelo modelo alvo, enquanto no cenário alvo, ocorre a validação dos modelos alvos contra amostras limpas (sem ataque) e adversárias. Além disso, no cenário alvo, será implementado o treinamento adversário para permitir a comparação de desempenho quando os modelos não possuem defesas contra os ataques. Dessa forma, é possível avaliar o desempenho dos modelos em condições adversas e como eles se comportam diante de possíveis ataques. A Random Forest estará presente apenas no cenário alvo. Por se tratar de um algoritmo mais robusto, será possível analisar seu desempenho contra amostras adversárias e compará-lo com as redes neurais. Podemos interpretar o cenário alvo como nosso detector de malware, enquanto o cenário de ataque pode ser visto como uma pessoa mal intencionada com o objetivo de burlar esse sistema.

O atacante tem o objetivo de manipular os malwares de forma que o detector seja incapaz de classificá-lo como um malware de fato. Para atingir esse objetivo, o atacante precisa simular um detector de malware e a partir disso gerar uma versão modificada do malware capaz de evadir a classificação no modelo alvo. Esse processo pode ser descrito em alguns passos:

- **1º passo - Criação do conjunto de dados**: o atacante precisa obter aplicativos benignos e maliciosos para a criação de um conjunto de dados. Sites como o VirusShare[2] disponibilizam diversos arquivos, incluindo aplicativos, que podem ser utilizados para conseguir atingir esse objetivo.

[2] Disponível em: https://virusshare.com/

- **2º passo - Treinamento do modelo**: com o conjunto de dados obtido, o atacante realiza a extração dos atributos por meio de diversas técnicas disponíveis e então treina uma rede neural que irá simular o detector alvo.
- **3º passo - Geração da versão modificada do malware**: a partir deste ponto o atacante pode utilizar as técnicas do AML e gerar uma versão modificada do seu próprio malware com a adição de novos atributos (chamadas API, permissões, etc.) que induzem o erro no classificador. Com o modelo da rede neural simulando o comportamento do detector, o atacante explora as vulnerabilidades do modelo, descobre quais atributos causam a evasão da classificação e, em seguida, modifica o malware de forma que ele seja classificado como benigno. A técnica de *transferability*, mencionado na Seção 2.3.1, pode permitir que essas falhas também aconteçam no modelo alvo, mesmo se o atacante não tiver conhecimento sobre o modelo utilizado.
- **4º passo - Propagação do malware**: com a versão modificada do malware, o atacante realiza a distribuição do aplicativo. Neste momento, o atacante pode ter um sistema específico como alvo ou pode tentar atingir diversos sistemas indiscriminadamente. A ideia é que a partir deste ponto alguns sistemas de detecção falhem ao classificar o malware e permitam que ele atue como um aplicativo benigno.

Os atributos que são adicionados ao malware não interferem em sua funcionalidade. Como nosso objetivo é apenas alterar informações coletadas do manifesto e do código binário dex, as informações adicionais serão na maioria das vezes ociosas, ou seja, não produzem efeitos no funcionamento do malware. Essas alterações existem apenas para induzir o erro do modelo.

Neste experimento, cada cenário é composto por um conjunto de modelos, os quais incluem redes neurais e uma Random Forest. As redes neurais serão construídas com diferentes arquiteturas, variando a quantidade de camadas entre 1 e 4, e a quantidade de neurônios por camada, com opções de 200, 500, 1000 e 5000 neurônios. Essa variedade de arquiteturas possibilita avaliar como diferentes modelos se comportam em cada cenário. Já a Random Forest estará presente apenas no cenário alvo. Uma vez que se trata de um algoritmo com características mais robustas, faremos uma comparação de desempenho entre a Random Forest e as rede neurais quando se deparam com amostras adversárias.

A Figura 11 resume o processo do experimento proposto. Cada seta saindo de um modelo M do cenário de ataque equivale a cinco tipos de ataques diferentes. Iremos avaliar a severidade dos cinco ataques contra cada um dos modelos no cenário alvo. O cenário alvo será composto por 11 modelos, sendo 5 deles, redes neurais com arquiteturas diferentes e que não passaram pelo treinamento adversário, e as mesmas 5 redes retreinadas, mas com o treinamento adversário, compondo assim o grupo de modelos com defesa. Por último, temos a Random Forest sem treinamento adversário. O cenário de ataque terá 5 modelos,

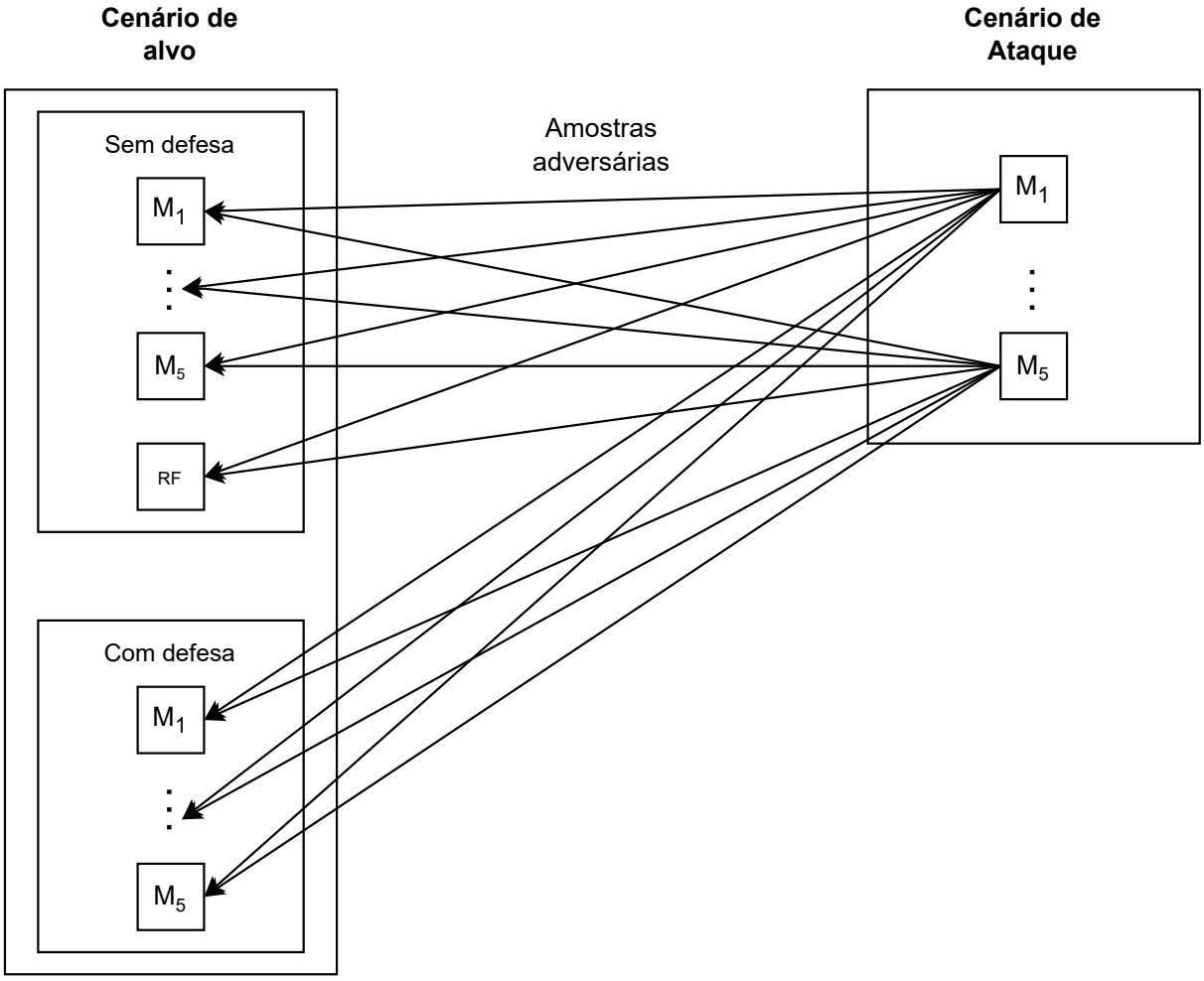

Figura 11 – Organização dos cenários com seus respectivos modelos. As redes neurais são nomeadas como M_1 até M_5 enquanto a Random Forest recebe a sigla RF.

que são usados para a geração das amostras adversárias. Os processos para cada etapa do experimento serão discutidos nas próximas seções

3.2.1 Treinamento dos modelos

O treinamento dos modelos, tanto das redes neurais quanto 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 para evitar viés e garantir a aleatoriedade do conjunto de treinamento. Já na fase de inferência, utilizaremos a proporção de 24:1 mencionada na Seção 3.1. 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.

Devido à natureza do problema, a quantidade de atributos (informações coletadas do manifesto e código) será determinada pelos arquivos dos aplicativos a partir dos quais

	Benignos	**Maliciosos**
1º Q.	23	35
Média	48	62
3º Q.	61	83
Max	9661	666

Tabela 2 – Estatística sobre a quantidade de atributos para cada aplicativo no conjunto de dados. A letra 'Q' indica quartil. Adaptado de Grosse *et al.* [2]

será gerado o conjunto de dados de treinamento. Como explicado por Pendlebury *et al.* [32], em um cenário real, os atributos conhecidos estão limitados à época em que os dados foram coletados. Em uma área em constante evolução, esse é um problema a ser entendido e esperado. Novos programas, com novos atributos e novas técnicas são produzidos o todo tempo. No caso deste trabalho, estamos utilizando aplicativos baseados no sistema operacional *Android* que possui informações bem definidas em sua documentação. Entretanto, para simular este comportamento apresentado, nossos modelos terão conhecimento apenas dos atributos presentes nos aplicativos durante a fase de treinamento.

Considerando as 4.000 amostras selecionadas, temos 25.550 atributos no total. Eles são obtidos pela união dos atributos de cada amostra. Nesse caso, cada aplicativo (amostra) possui um conjunto de atributos extraídos. Observando a Tabela 2 vemos que a média de atributos por aplicativo é de 55. Sendo assim, os vetores de atributos, Figura 8, se tornam esparsos visto que para os 25.550 atributos apenas 55, em média, estarão presentes. Isso é algo a ser observado durante as avaliações do desempenho dos modelos. O agrupamento de todos os atributos do nosso conjunto de dados será frequentemente referido como cabeçalho, pois serve como referência para a posição de cada atributo no vetor de atributos.

A Figura 12 resume de forma clara todo o processo ao qual os modelos são submetidos. Durante a fase de treinamento, o cabeçalho a ser utilizado é obtido e o modelo é treinado com as amostras disponíveis. O cabeçalho gerado neste processo é posteriormente utilizado na fase de inferência, onde novas amostras são aplicadas no modelo para testar sua capacidade de generalização - ou seja, verificar se ele é capaz de aplicar o conhecimento adquirido durante o treinamento em amostras que ele nunca viu antes. É nesta fase que avaliamos o desempenho do modelo de aprendizado de máquina.

3.2.1.1 Implementação das Redes neurais

As redes neurais são construídas utilizando a biblioteca *Pytorch* [3] na versão 2.0. A rede neural é construída de forma simples, para cada neurônio nas camadas ocultas atribuímos uma função ativação conhecida como ReLu (*rectifier linear unit*). Na camada de saída do modelo, utilizamos a função de ativação conhecida como LogSoftmax. Essa

[3] Disponível em: https://pytorch.org/

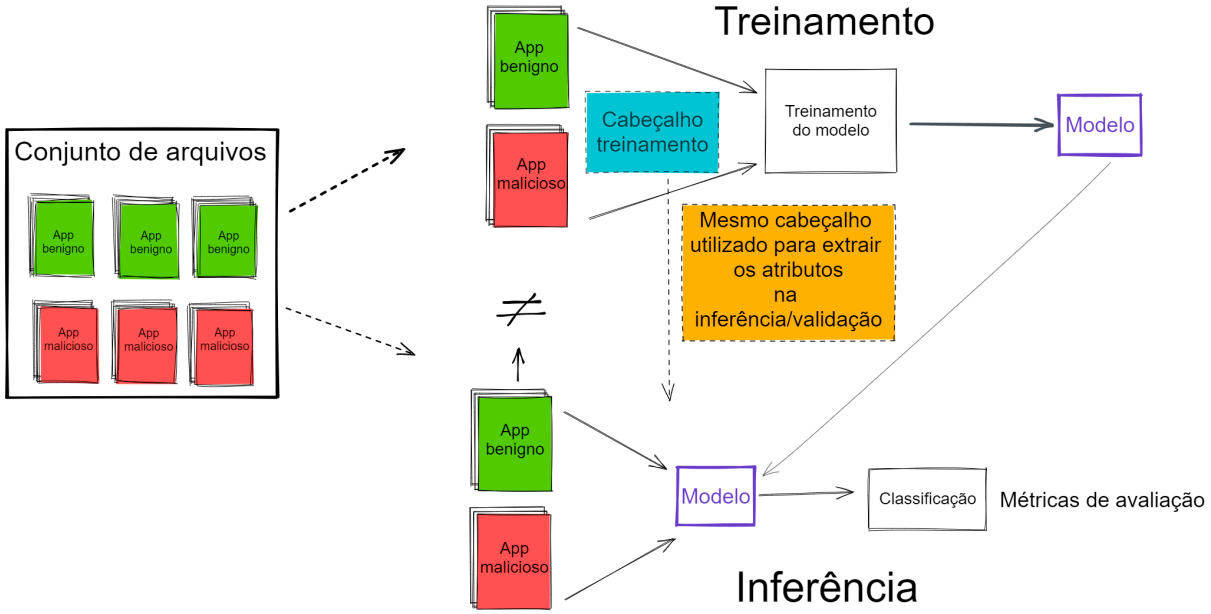

Figura 12 – Exemplo do processo de treinamento e inferência dos modelos de aprendizado de máquina.

função gera um vetor de probabilidades na saída do modelo. As definições de ambas funções podem ser vistas nas Equações 3.1 e 3.2, respectivamente

$$f(x) = max(0, x) \tag{3.1}$$

$$LogSoftmax(x_i) = log(\frac{\exp(x_i)}{\sum_j \exp(x_j)}) \tag{3.2}$$

A ReLu é uma função não-linear que retorna x caso positivo, e 0, caso contrário. Já a LogSoftmax, é aplicada em um vetor de valores reais, transformado-os em uma distribuição de probabilidade normalizada, onde a soma de todas as probabilidades é igual a 1. Na Equação 3.2, x_i representa o i-ésima classe do nosso sistema, e j varia de 1 até a quantidade de classes no sistema.

A função de custo dos modelos é a *CrossEntropyLoss*, e utilizamos o algoritmo Adam para otimizá-la. Esse algoritmo é uma versão estendida do *stochastic gradient descent*. 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.

3.2.1.2 Implementação da Random Forest

A implementação da Random Forest foi feita utilizando a biblioteca *Scikit-learn*[4] [34]. O algoritmo é composto de 100 árvores e utilizamos o critério Gini para medir a qualidade das divisões. A quantidade máxima de atributos para garantir o melhor divisão a dada pela raiz quadrada do total de atributos. Utilizamos a opção de *bootstrap* como verdadeira permitindo que o algoritmo crie amostras aleatórias na construção das árvores.

3.2.2 Métodos para geração de amostras adversárias

Neste trabalho serão utilizados cinco métodos diferentes para gerar as amostras adversárias: rFGSM, dFGSM, BGA, BCA e o método de Grosse. O objetivo de cada método é realizar ataques de evasão manipulando apenas amostras que foram classificadas como maliciosas. Os métodos possuem algumas características similares: eles dependem de um modelo de classificação pré-treinado para gerar as amostras e suas alterações não interferem na funcionalidade do malware.

Os ataques dFGSM e rFGSM são variações do FGSM mencionado na seção 2.3.1. No entanto, neste caso, estamos lidando com um conjunto discreto, diferente do abordado pelo FGSM, onde o ruído δ obtido não é aplicável a uma amostra binária (vetor de atributos), como é o caso deste trabalho. Para contornar esse problema, o dFGSM e o rFGSM estabelecem critérios de arredondamento. No dFGSM, a letra 'd' significa determinístico, ou seja, um limiar é definido para determinar se um atributo deve ser adicionado ou não ao vetor de atributos binário. Já no rFGSM, a letra 'r' significa randômico, ou seja, o limiar, nesse caso, é randômico. Ambos os ataques possuem grande impacto nas amostras. O ruído encontrado pelo FGSM é aplicado à amostra toda, sendo assim, a depender do ϵ escolhido, podemos ter um ataque mais ou menos agressivo, sendo $\epsilon = 1$ o valor máximo.

O método de Grosse é um ataque mais direcionado que visa alterar poucos atributos. O ataque tenta estabelecer uma relação do atributo com a saída do modelo. Para isso é utilizado uma matriz Jacobiana. A matriz é composta pela derivada parcial dos atributos em relação à saída do modelo, podendo ser definida da seguinte maneira:

$$J_f = \left[\frac{\partial F_i(X)}{\partial X_j}\right]_{i\epsilon 0,1; j\epsilon[1,m]} \tag{3.3}$$

Na Equação 3.3, temo que F representa um modelo treinado e $F_i(X)$ é a saída do modelo em relação à classe i. No nosso caso, i pode ser 0 ou 1, onde 0 é classe de benignos e 1 a classe de amostras maliciosas. X_j indica o atributo verificado na posição j que varia de 1 até m, sendo m a quantidade total de atributos no vetor (25.550). Com isso, o método escolhe o atributo que maximiza o erro do modelo em relação à classe escolhida,

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

no caso, a classe de benignos (classe 0). Esse processo é repetido até que uma quantidade máxima k de atributos seja modificada. Neste trabalho, tomamos $k = 20$, pois, admite uma quantidade pequena de atributos modificados e demonstra ter bastante impacto na classificação, como demonstrado por Grosse *et al.* [24], Shaukat *et al.* [1] e Al-Dujaili *et al.* [31].

Os ataques BGA e BCA foram elaborados especificamente para o problema de malwares, ou seja, lidam com os vetores de atributos binários. O BGA (*Bit Gradient Ascent*) modifica o atributo na j-ésima posição caso a derivada parcial contribua mais ou igualmente para a norma ℓ_2 (distância Euclidiana) do gradiente em comparação com o restante dos atributos. Já o BCA (*Bit Coordinate Ascent*), semelhante ao método de Grosse, atualiza um atributo por iteração para maximizar o erro do modelo. Um atributo só será escolhido caso ele contribua mais para a perda do modelo. Neste caso, utilizamos a quantidade de iterações do BCA sendo a mesma do método de Grosse, ou seja, $k = 20$.

Os ataques dFGSM, rFGSM e BGA são frequentemente representados como dFGSM^k, rFGSM^k e BGA^k. O k indica que eles são ataques de k iterações. Isso significa que o método é aplicado k vezes sobre a mesma amostra, ou seja, na primeira iteração a amostra ainda não foi alterada, na segunda iteração, o ataque é reaplicado sobre a amostra já alterada pela primeira iteração e assim sucessivamente. O valor de k foi fixado em 50 iterações para todos esses ataques. Por simplicidade, iremos utilizar os nomes dos métodos sem a informação k.

A utilização desses ataques irá nos permitir entender alguns pontos importantes. O dFGSM, rFGSM e o BGA são ataques de alta intensidade, já que eles alteram uma quantidade significativa de atributos. A quantidade de atributos manipulados pelo ataque pode torná-lo mais suscetível a detecção e invalidá-lo antes mesmo dele acontecer. Já os métodos BCA e Grosse possuem objetivos semelhantes, alterar a menor quantidade de atributos causando o maior impacto possível no classificador.

A implementação dos ataques foi baseada no código fornecido por Al-Dujaili *et al.* [31], que está disponível em seu repositório.

3.2.3 Defesa baseada em treinamento adversário

O treinamento adversário é um processo teórico simples. Durante a fase de treinamento do modelo alvo, é apresentada uma quantidade de amostras maliciosas ao modelo. A intenção é que as particularidades do ataque possam ser generalizadas pelo modelo e ele passe a se proteger contra esses ataques.

Neste estudo, foram selecionados aleatoriamente 200 malwares que não haviam sido selecionados nem para o treinamento e nem para a fase de inferência. Para cada um dos ataques utilizados nesse trabalhos, iremos gerar 200 amostras adversárias a partir

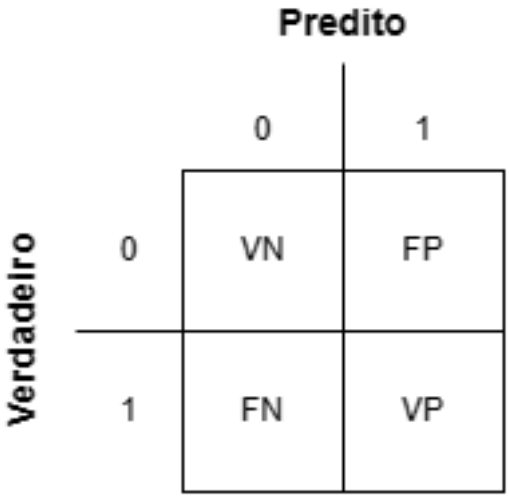

Figura 13 – Exemplo de uma matriz de confusão.

desses dados selecionados, somando 1.000 novas amostras no total. Em seguida, utilizando o mesmo processo, iremos retreinar o nosso modelo. Entretanto, para mantermos o equilíbrio das classes, acrescentamos aleatoriamente 1.000 amostras benignas. Em suma, o treinamento irá contar com as 4.000 amostras iniciais do treinamento e iremos adicionar mais 2.000 amostras, sendo 1.000 amostras adversárias e outras 1.000 amostras benignas, totalizando 6.000 amostras durante a fase de treinamento.

Durante esse processo, mantivemos os 25.550 atributos mencionados na Seção 3.2.1. Para gerar as amostras adversárias utilizamos a versão sem defesa do modelo. Por exemplo, para treinar o modelo M_1 com defesa, utilizamos o M_1 sem defesa para gerar as amostras.

3.2.4 Métricas de avaliação

A avaliação do desempenho dos modelos será realizada utilizando informações coletadas da matriz de confusão. Essa matriz relaciona os valores verdadeiros de determinada amostra com os que foram preditos pelo modelo. A Figura 13 é uma representação visual dessa matriz. Para um problema binário, temos as predições negativas, 0, e as predições positivas, 1.

A matriz de confusão organiza esses dados e agrega informação sobre a classificação. Por exemplo, quando uma amostra é negativa (benigna), mas o modelo prediz uma amostra como positiva (maliciosa), temos um falso positivo (FP). Se a amostra for negativa, e o modelo a prediz como negativa, temos um acerto, logo, temos um verdadeiro negativo (VN). A mesma análise ocorrerá para as amostras positivas. Caso uma amostra positiva seja classificada como negativa, temos um falso negativo (FN). Se uma amostra for positiva e sua classificação também for positiva, temos um verdadeiro positivo (VP).

Essas informações dão indícios de como está o desempenho do nosso modelo. O ideal é que o erro seja o menor possível para o problema. Ou seja, queremos que os negativos ocorram com menos frequência. Entretanto, essa é uma tarefa difícil que depende do problema em questão. No nosso caso, a preocupação é maior em relação aos falsos negativos. Os falsos negativos indicam que malwares estão sendo classificados como benignos. Essa situação causa insegurança sobre a capacidade de um detector de malware.

Efetivamente, é necessário que possamos entender melhor o desempenho do modelo. Analisar apenas a quantidade de falsos negativos pode causar uma falsa impressão de segurança. Para isso, existem algumas métricas baseadas nas informações fornecidas pela matriz de confusão. Algumas dessas são: a acurácia, precisão, sensibilidade e o *F1-score*.

A acurácia nos mostra a quantidade de acertos do modelo em relação ao total de amostras classificadas. A precisão observa a quantidade de falsos positivos em relação a todas amostras classificadas como positivas. A sensibilidade mede a proporção de acertos ao classificar amostras positivas. Já o F1-score é uma média harmônica entre a precisão e sensibilidade. As Equações 3.4, 3.5, 3.6 e 3.7, demonstram como esses cálculos podem ser obtidos a partir da matriz de confusão.

$$Acurácia = \frac{VP + VN}{VP + VN + FN + FP} \tag{3.4}$$

$$Precisão = \frac{VP}{VP + FP} \tag{3.5}$$

$$Sensibilidade = \frac{VP}{VP + FN} \tag{3.6}$$

$$F1 - score = 2 * \frac{Precisão * Sensibilidade}{Precisão + Sensibilidade} \tag{3.7}$$

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 que mede a severidade do ataque (SA) contra o modelo. Ela estabelece uma razão entre a sensibilidade antes e depois do ataque. A Equação 3.8 demonstra como podemos calcular a severidade do ataque.

$$SA = 1 - \frac{Sensibilidade(Depois)}{Sensibilidade(Antes)} \tag{3.8}$$

4 RESULTADOS

Após definir os cenários e o experimento, iniciamos a seleção dos 5 modelos de redes neurais para compor ambos os cenários, tendo em conta o seu desempenho medido pelo *F1-score*. Foram criados 30 modelos com arquiteturas diferentes, variando a quantidade de camadas e neurônios por camada, e apenas os que obtiveram os maiores *F1-scores* foram escolhidos. A Tabela 9 apresenta todos os modelos e suas respectivas arquiteturas. Já a Tabela 3 destaca os 5 modelos escolhidos para os cenários, tendo o modelo M_1 alcançado o maior *F1-score* com $0,64$, enquanto o 5^o modelo, M_5, obteve um *F1-score* de $0,59$. Uma característica semelhante em todos os modelos é a baixa precisão. Nesse caso, isso indica que nossos modelos erram mais ao classificar amostras benignas, causando falsos positivos.

A Tabela 4 nos mostra o treinamento adversário realizado sobre cada modelo da Tabela 3. Podemos observar que a precisão dos modelos nesse caso aumentou, sendo a precisão mais baixa a do modelo M_4, com $precisão = 0,50$, enquanto a maior precisão observada foi a do modelo M_3, $precisão = 0,69$. Porém, todos os modelos apresentaram uma queda na sensibilidade de aproximadamente $0,04$ pontos. Isso significa que os modelos erraram mais ao classificar amostras maliciosas. Apesar da troca entre a precisão e a sensibilidade durante o treinamento adversário, os modelos apresentaram um *F1-score* mais robusto, sendo o menor deles de $0,64$ obtido pelo modelo M_4 e o maior de $0,77$ pelo modelo M_3.

No geral, os modelos se comportaram de maneira semelhante, independente da arquitetura. Os dados de treinamento foram os mesmos durante o processo, assim como os dados da fase de inferência. Não foi possível apontar uma melhora dos modelos em relação ao número de camadas ou a quantidade de neurônios.

Nome	**Modelo**	**Acurácia**	**Sensibilidade**	**Precisão**	**F1-Score**
M_1	5000x1000x500x200	0,960	0,910	0,497	0,643
M_2	5000x200	0,957	0,940	0,482	0,637
M_3	1000x1000x1000	0,956	0,940	0,472	0,629
M_4	200x500x200	0,952	0,930	0,451	0,608
M_5	200x200x200	0,948	0,940	0,433	0,593

Tabela 3 – Modelos da rede neural utilizados no experimento com suas respectivas métricas

Modelos com defesa (Treinamento Adversário)					
Nome	**Modelo**	**Acurácia**	**Sensibilidade**	**Precisão**	**F1-Score**
M_1	5000x1000x500x200	0,973	0,870	0,617	0,722
M_2	5000x200	0,975	0,880	0,638	0,739
M_3	1000x1000x1000	0,980	0,880	0,693	0,775
M_4	200x500x200	0,960	0,920	0,500	0,648
M_5	200x200x200	0,962	0,910	0,511	0,655

Tabela 4 – Modelos da rede neural retreinados utilizando o treinamento adversário

4.1 Desempenho das redes neurais nos cenários de ataque e defesa

As Tabelas 5 e 6 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 arquitetura. Cada linha da tabela representa a média da severidade do ataque considerando os 5 modelos (M_1-M_5). As colunas representam as arquiteturas utilizadas pelos atacantes para gerar as amostras adversárias.

A diferença na arquitetura dos modelos não proporcionou uma maior resistência aos ataques nem possibilitou ataques mais efetivos. A ideia era que redes mais complexas poderiam se tornar mais resistentes aos ataques, mas isso não foi observado como um ponto significativo neste experimento. Não foi possível encontrar uma arquitetura que se destacasse claramente como mais ou menos robusta em relação às outras. Em resumo, os modelos tiveram um comportamento semelhante independente da arquitetura utilizada. Na sequência, discutiremos os resultados em termos da severidade dos ataques e da eficácia da defesa.

Em média, os ataques alcançaram impactos consideráveis contra os modelos. Observando a Tabela 5, vemos os modelos quando não possuem defesa. Os ataques dFGSM, rFGSM e BGA obtiveram severidades máximas ($severidade = 1$) contra todos os modelos alvos utilizando pelo menos um modelo de ataque específico para gerar as amostras. Neste caso, a severidade indica que os ataques conseguiram transformar todas as amostras maliciosas em benignas. O método de Grosse teve um desempenho muito próximo dos ataques mencionados acima. A severidade média mais alta obtida por Grosse foi de $0,978$ e a mais baixa de $0,959$. O ataque que obteve menos sucesso nesse cenário foi o BCA. Quando utilizando os modelos M_1, M_2 e M_4 sua severidade média ficou menor que $0,650$, alcançando seu máximo apenas com a arquitetura M_3, quando obteve $0,943$ de severidade.

Na Tabela 6, onde os modelos alvo receberam treinamento adversário, observamos uma diminuição aguda da severidade do ataque com os métodos BCA e Grosse. A severidade média máxima obtida por Grosse contra os modelos defendidos foi de $0,218$,

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

Tabela 5 – 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 (M_1-M_5) e as colunas mostram qual a arquitetura utilizada pelo atacante.

uma diminuição de $0,76$ na severidade do ataque. Já o BCA conseguiu uma severidade média máxima de $0,541$ com uma variação de $\pm 0,12$. Uma diminuição da severidade em $0,40$ em relação ao máximo atingido contra os modelos alvo sem defesa. Por outro lado, os ataques dFGSM, rFGSM e BGA continuaram apresentando uma elevada severidade, atingindo novamente valores máximos. No caso das variações do FGSM, apenas o ataque gerado pelo modelo M_5 sofreu uma queda de $0,20$. Nesse caso, não ficou claro qual a influência da arquitetura na diminuição da severidade do ataque.

Uma características dos ataques dFGSM, rFGSM e BGA é a quantidade de atributos que foram manipulados durante o processo de geração de amostras maliciosas. Enquanto limitamos o BCA e o Grosse a alterar cerca de 20 atributos no máximo, esses métodos alvejam uma quantidade significativa de atributos. Isso permite que esses ataques possam explorar ainda mais os limites dos modelos alvo, uma vez que mesmo os modelos treinados com defesa adversária não foram capazes de evitar impactos expressivos causados por esses ataques contra os classificadores.

O que pode influenciar uma defesa mais efetiva seria encontrar a quantidade certa de amostras durante a fase de treinamento. Neste experimento, utilizamos 200 amostras maliciosas geradas por cada ataque. Aparentemente, essa quantidade foi suficiente apenas contra o BCA e Grosse. Para os ataques de maior intensidade, essa quantidade não se mostrou significativa.

Em suma, os ataques de maior intensidade como o dFGSM, rFGSM e BGA tiveram severidades maiores contra os modelos sem defesa e com defesa. Eles foram capazes de obter severidades máximas e não foram afetados pelo treinamento adversário. Em contraste, os ataques de baixa intensidade, como o BCA e Grosse, foram efetivos contra os modelos sem defesa, entretanto, tiveram pouco impacto contra os modelos com defesa.

O treinamento adversário mostrou-se resiliente contra os ataques de baixa intensidade, mas não foi capaz de generalizar adequadamente para todos os tipos de ataques utilizados neste experimento. De acordo com Grosse *et al.* [2], ao aplicarmos o treinamento adversário, nossos modelos deveriam obter uma boa generalização e se tornarem mais resilientes a amostras adversárias.

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

Tabela 6 – 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 (M_1-M_5) e as colunas mostram qual a arquitetura utilizada pelo atacante.

	Acurácia	**Sensibilidade**	**Precisão**	**F1-score**
Random Forest (sem defesa)	0,956	0,920	0,472	0,624

Tabela 7 – Métricas da Random Forest sem mecanismo de defesa.

4.2 Comparação entre a Random Forest e as redes neurais

Os resultados das métricas da Random Forest foram comparáveis aos das redes neurais. Conforme demonstrado na Tabela 7, o *F1-score* da Random Forest foi de 0,62, ficando apenas 0,02 pontos abaixo do modelo M_1 sem defesa. De maneira semelhante às redes neurais, a Random Forest também apresentou baixa precisão, indicando que a alta taxa de falso positivos parece fazer parte do problema analisado.

A Tabela 8 nos mostra a severidade dos ataques contra a Random Forest. Ela foi construída utilizando a mesma ideia das Tabelas 5 e 6 explicadas no início da Seção 4.1. Entretanto, nesse caso, os ataques são realizados apenas contra a Random Forest, então não utilizamos uma média para obter a severidade dos ataques.

Ao analisarmos a Tabela 8, podemos notar uma pequena variação nos resultados dos ataques em relação à arquitetura utilizada. Alguns casos, como o das arquiteturas dos modelos M_1 e M_5, parecem destoar da tendência geral, especialmente quando utilizadas para gerar amostras adversárias com os métodos dFGSM e rFGSM. É importante destacar que o método rFGSM, que se utiliza de manipulações baseadas em arredondamentos aleatórios, apresentou menor impacto na Random Forest em comparação com as 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, como o BCA e Grosse. Parte disso pode ser explicado pela forma como o algoritmo funciona. A Random Forest emprega 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, aumentando a diversidade

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 8 – Severidade dos ataques contra a Random Forest. As linhas representam a severidade do ataque contra a Random Forest e as colunas mostram qual a arquitetura utilizada pelo atacante.

do modelo e reduzindo a chance de *overfitting* [35] [36]. 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.

4.3 Discussão

A partir dos resultados encontrados, foi possível sintetizar as informações observadas e responder as perguntas que guiaram o experimento proposto.

- **Qual a severidade dos ataques?**

  Os ataques de alta intensidade como o dFGSM, rFGSM e BGA causaram o maior impacto contra os detectores no geral. Tanto as redes neurais, sem e com defesa, quanto a Random Forest, não foram robustos contra as amostras geradas por esses ataques. Em diversas situações, a severidade desses ataques foi máxima, ou seja, $SA = 1$. Já os ataques de baixa intensidade (BCA e Grosse) foram efetivos apenas contra as redes neurais sem defesa, obtendo, em algumas situações, severidade máxima.

  Nesse contexto, apesar do impacto causado pelos ataques de alta intensidade, eles são mais suscetíveis a serem detectados por conta da quantidade de atributos ociosos presentes nos aplicativos. Mas, por outro lado, os ataques de baixa intensidade alteram poucos atributos e conseguiram ser efetivos contra as redes neurais sem defesa.

- **Qual a eficácia do mecanismo de defesa?**

  O treinamento adversário teve impacto apenas contra as amostras adversárias geradas pelos ataques de baixa intensidade. Os ataques de alta intensidade ainda foram efetivos contra os modelos que aplicaram a técnica de defesa. Isso indica que a quantidade de amostras utilizadas no treinamento pode não ter sido suficiente para combater os ataques de alta intensidades, entretanto, foi efetiva contra os ataques de baixa intensidade, que são ataques mais difíceis de serem detectados.

- **Qual a influência da arquitetura do modelo no ataque e defesa?**

  Durante o experimento não foram encontrados indícios de que a arquitetura das redes aumentava ou diminuía a robustez ou impacto dos ataques. Apesar de alguns casos isolados, não foi observado um padrão que se repetia em determinada arquitetura. Essas observações ocorreram tanto para os modelos alvos quanto para os modelos utilizados pelo atacante.

- **Qual o desempenho da Random Forest comparada às redes neurais?**

  Apesar do treinamento adversário não ter sido aplicado à Random Forest, ela conseguiu anular todos os ataques de baixa intensidade. Nesse sentido, a Random Forest se mostrou um algoritmo mais robusto em comparação com as redes neurais.

5 CONCLUSÃO

A detecção de malware é uma tarefa complexa e desafiadora, especialmente considerando o cenário de ameaças em constante evolução. Técnicas de aprendizado de máquina têm sido amplamente utilizadas para a detecção de malwares, mas, como mostrado em vários estudos, esses modelos são suscetíveis a amostras adversárias. Portanto, explorar maneiras de se defender contra esses ataques se torna um tema importante na área de detecção de malware.

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 classificadores de malwares. Foi observado que os classificadores 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 ataques 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.

Como trabalhos futuros, há alguns pontos que poderiam ser explorados. Um deles é aprimorar a robustez dos modelos de classificação de malwares, utilizando técnicas de redução de atributos e investigar como esses modelos se comportam em relação a ataques de alta e baixa intensidade. Outro ponto é aprofundar o entendimento sobre a efetividade do treinamento adversário contra amostras adversárias. Compreender melhor as particularidades do treinamento adversário pode ajudar a criar modelos mais robustos e, consequentemente, mais eficientes na detecção de malwares.

REFERÊNCIAS

[1] SHAUKAT, K.; LUO, S.; VARADHARAJAN, V. A novel method for improving the robustness of deep learning-based malware detectors against adversarial attacks. *Engineering Applications of Artificial Intelligence*, Elsevier, v. 116, p. 105461, 2022.

[2] GROSSE, K. et al. Adversarial perturbations against deep neural networks for malware classification. *arXiv preprint arXiv:1606.04435*, 2016.

[3] ASLAN, Ö. A.; SAMET, R. A comprehensive review on malware detection approaches. *IEEE Access*, IEEE, v. 8, p. 6249–6271, 2020.

[4] CHRISTODORESCU, M. et al. Semantics-aware malware detection. In: IEEE. *2005 IEEE symposium on security and privacy (S&P'05)*. [S.l.], 2005. p. 32–46.

[5] YE, Y. et al. A survey on malware detection using data mining techniques. *ACM Computing Surveys (CSUR)*, ACM New York, NY, USA, v. 50, n. 3, p. 1–40, 2017.

[6] BAZRAFSHAN, Z. et al. A survey on heuristic malware detection techniques. In: IEEE. *The 5th Conference on Information and Knowledge Technology*. [S.l.], 2013. p. 113–120.

[7] PAPERNOT, N. et al. Practical black-box attacks against machine learning. In: *Proceedings of the 2017 ACM on Asia conference on computer and communications security*. [S.l.: s.n.], 2017. p. 506–519.

[8] KOLOSNJAJI, B. et al. Adversarial malware binaries: Evading deep learning for malware detection in executables. In: IEEE. *2018 26th European signal processing conference (EUSIPCO)*. [S.l.], 2018. p. 533–537.

[9] GANDOTRA, E.; BANSAL, D.; SOFAT, S. Malware analysis and classification: A survey. *Journal of Information Security*, Scientific Research Publishing, v. 2014, 2014.

[10] MITCHELL, T. M.; MITCHELL, T. M. *Machine learning*. [S.l.]: McGraw-hill New York, 1997. v. 1.

[11] SAMUEL, A. L. Some studies in machine learning using the game of checkers. ii—recent progress. *IBM Journal of research and development*, IBM, v. 11, n. 6, p. 601–617, 1967.

[12] ZHOU, Z.-H. *Machine learning*. [S.l.]: Springer Nature, 2021.

[13] REDES neurais. <https://www.ibm.com/br-pt/cloud/learn/neural-networks>. Acessado: 2022-08-15.

[14] DEEP Learning Book. <https://www.deeplearningbook.com.br/funcao-de-ativacao/#:~:text=As%20fun%C3%A7%C3%B5es%20de%20ativa%C3%A7%C3%A3o%20s%C3%A3o,fornecida%20ou%20deve%20ser%20ignorada.> Acessado: 2022-09-07.

[15] BREIMAN, L. Random forests. *Machine learning*, Springer, v. 45, n. 1, p. 5–32, 2001.

[16] DECISION Trees. <https://scikit-learn.org/stable/modules/tree.html#mathematical-formulation>.

[17] BIAU, G.; SCORNET, E. A random forest guided tour. *Test*, Springer, v. 25, n. 2, p. 197–227, 2016.

[18] NG, A. Cs229 lecture notes. *CS229 Lecture notes*, v. 1, n. 1, p. 1–3, 2000.

[19] LECTURE 9: SVM. <https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote09.html>.

[20] SUPPORT Vector Machines. <https://scikit-learn.org/stable/modules/svm.html>.

[21] HUANG, L. et al. Adversarial machine learning. In: *Proceedings of the 4th ACM workshop on Security and artificial intelligence*. [S.l.: s.n.], 2011. p. 43–58.

[22] GOODFELLOW, I. J.; SHLENS, J.; SZEGEDY, C. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[23] YANG, L. et al. Jigsaw puzzle: Selective backdoor attack to subvert malware classifiers. *arXiv preprint arXiv:2202.05470*, 2022.

[24] GROSSE, K. et al. Adversarial examples for malware detection. In: SPRINGER. *European symposium on research in computer security*. [S.l.], 2017. p. 62–79.

[25] PAPERNOT, N. et al. The limitations of deep learning in adversarial settings. In: IEEE. *2016 IEEE European symposium on security and privacy (EuroS&P)*. [S.l.], 2016. p. 372–387.

[26] PIERAZZI, F. et al. Intriguing properties of adversarial ml attacks in the problem space. In: IEEE. *2020 IEEE symposium on security and privacy (SP)*. [S.l.], 2020. p. 1332–1349.

[27] LABACA-CASTRO, R. et al. Realizable universal adversarial perturbations for malware. ArXiv, 2021.

[28] PAPERNOT, N. et al. Distillation as a defense to adversarial perturbations against deep neural networks. In: IEEE. *2016 IEEE symposium on security and privacy (SP)*. [S.l.], 2016. p. 582–597.

[29] SZEGEDY, C. et al. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[30] ROSENBERG, I. et al. Generic black-box end-to-end attack against state of the art api call based malware classifiers. In: SPRINGER. *International Symposium on Research in Attacks, Intrusions, and Defenses*. [S.l.], 2018. p. 490–510.

[31] AL-DUJAILI, A. et al. Adversarial deep learning for robust detection of binary encoded malware. In: IEEE. *2018 IEEE Security and Privacy Workshops (SPW)*. [S.l.], 2018. p. 76–82.

[32] PENDLEBURY, F. et al. {TESSERACT}: Eliminating experimental bias in malware classification across space and time. In: *28th USENIX Security Symposium (USENIX Security 19)*. [S.l.: s.n.], 2019. p. 729–746.

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

[34] PEDREGOSA, F. et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, v. 12, p. 2825–2830, 2011.

[35] SCITKIT-LEARN. *Ensemble methods*. Disponível em: <https://scikit-learn.org/stable/modules/ensemble.html#bagging-meta-estimator>.

[36] LTD, Q. *Bootstrap Aggregation, Random Forests and Boosted Trees*. Disponível em: <http://www.quantstart.com/articles/bootstrap-aggregation-random-forests-and-boosted-trees/>.

Apêndices

Modelo	Acurácia	Sensibilidade	Precisão	F1-Score
200	0,945	0,930	0,417	0,576
500	0,942	0,900	0,398	0,552
1000	0,942	0,910	0,401	0,557
5000	0,936	0,950	0,380	0,543
200x200	0,944	0,940	0,414	0,575
200x500	0,931	0,950	0,361	0,523
200x1000	0,943	0,930	0,406	0,565
200x5000	0,940	0,950	0,397	0,560
500x200	0,943	0,950	0,409	0,572
500x500	0,940	0,950	0,394	0,557
500x1000	0,913	0,950	0,308	0,466
500x5000	0,947	0,930	0,425	0,583
1000x200	0,946	0,940	0,422	0,582
1000x500	0,948	0,940	0,429	0,589
1000x1000	0,938	0,940	0,387	0,548
1000x5000	0,944	0,950	0,411	0,574
5000x200	**0,957**	**0,940**	**0,482**	**0,637**
5000x500	0,938	0,930	0,386	0,545
5000x1000	0,945	0,940	0,418	0,578
5000x5000	0,948	0,920	0,430	0,586
200x200x200	**0,948**	**0,940**	**0,433**	**0,593**
200x500x200	**0,952**	**0,930**	**0,451**	**0,608**
200x1000x200	0,948	0,940	0,433	0,593
200x5000x200	0,945	0,940	0,418	0,578
500x500x500	0,944	0,930	0,412	0,571
1000x1000x1000	**0,956**	**0,940**	**0,472**	**0,629**
5000x5000x5000	0,940	0,940	0,397	0,558
200x500x200x500	0,938	0,940	0,385	0,547
5000x1000x500x200	**0,960**	**0,910**	**0,497**	**0,643**
5000x1000x1000x5000	0,945	0,930	0,417	0,576

Tabela 9 – Tabela contendo 30 modelos com variações em suas arquiteturas. Os modelos em negrito são os 5 modelos que obtiveram os melhores *F1-score*