Avaliação de Abordagens para Classificação de Malware Bancário sob a Presença de Concept Drift

Anais do Computer on the Beach

Eloiza Rossetto dos Santos
eloiza.rossetto@ufpr.br
Departamento de Informática
Universidade Federal do Paraná
Curitiba, Paraná, Brasil

André Grégio
gregio@ufpr.br
Departamento de Informática
Universidade Federal do Paraná
Curitiba, Paraná, Brasil

Paulo Lisboa de Almeida
paulorla@ufpr.br
Departamento de Informática
Universidade Federal do Paraná
Curitiba, Paraná, Brasil

Abstract

Financial malware is an increasing threat due to the potential of profit for cybercriminals. Year after year, the arms race between malware developers and security professionals foster the release of millions of malware variants. Although machine learning techniques have been successfully applied to malware classification tasks, the occurrence of concept drifts requires constant updates (or even retraining) of these models. In this work, we evaluate how distinct machine learning training approaches for malware classification when we consider the arrival flow of samples as a data stream. We also observe the manifestation of concept drift in an actual dataset comprised of thousands of financial malware samples, discussing different incremental learning approaches to deal with a custom dataset and how concept drift can arise in malware data.

Keywords

Cybersecurity, Machine Learning, Dynamic Malware Analysis, Concept Drift, Natural Language Processing

1 Introdução

Nos últimos anos, o aumento significativo de ataques cibernéticos tem gerado prejuízos financeiros expressivos globalmente. Somente em 2022, foram investidos aproximadamente R$ 3,5 bilhões para prevenção de fraudes/segurança bancária online pelo sistema financeiro brasileiro [1]. Ainda assim, estima-se que golpes financeiros virtuais causaram um prejuízo de mais de R$ 500 milhões aos usuários brasileiros de serviços digitais em 2023 [2] e, entre junho de 2023 e julho de 2024, foram registrados 725 milhões de tentativas de infecção por *malware* no Brasil [3]. À medida que ciber-criminosos aprimoram seus programas maliciosos para torna-los cada vez mais complexos de serem detectados, a comunidade de cibersegurança cria novas tecnologias e estratégias para proteger sistemas interconectados, dispositivos e usuários. Uma das tecnologias que vem sendo utilizada amplamente no contexto dos mecanismos de defesa são os algoritmos de aprendizado de máquina, capazes de aprender e reconhecer padrões complexos a partir de dados coletados que representem o alvo.

Em geral, os algoritmos de aprendizado de máquina utilizam uma abordagem de aprendizado baseada em lotes, na qual um conjunto de dados fixo de amostras é repartido em dois subconjuntos: um para treinamento e outro para avaliação do modelo. Essa abordagem considera um cenário em que a distribuição dos dados é estática ao longo do tempo, porém tal cenário não é usualmento visto no escopo dos ataques por códigos maliciosos, que se disseminam por campanhas sazonais e com inúmeras variações de amostras para escapar de mecanismos antivírus. Exemplares de malware estão sempre evoluindo para evadir sistemas de defesa e se manterem atualizados contra mudanças em softwares atacados [4]. Tais variações para atualização e evasão fazem com que exemplares apresentem mutações constantes ao longo do tempo, caracterizando assim a atuação das diversas variantes que compõem uma família como uma série temporal e, consequentemente, tornando o paradigma de aprendizado em lotes inadequado para esse tipo de dado.

Além disso, essa natureza mutante de códigos maliciosos faz com que classificadores de aprendizado de máquina estejam suscetíveis aos efeitos de *concept drifts*. Um *concept drift* é caracterizado pela mudança das características de dados ao longo do tempo tornando classificadores anteriormente treinados obsoletos [5]. Ao trazermos à tona a identificação de *concept drifts* para conjuntos de dados da área de segurança computacional, observa-se uma escassez de trabalhos que discutam apropriadamente tal fenômeno relacionado à detecção de códigos maliciosos. Para lidar com esse ambiente dinâmico em que os malwares são caracterizados, o presente trabalho propõe lidar com a classificação de exemplares de malware de forma a considerar cada família como um fluxo (*stream*) de dados, ao invés de lotes de dados como é feito usualmente. Com isso, pode-se tirar vantagem do paradigma de aprendizado online (também chamado de incremental), o qual apresenta algoritmos capazes de se adaptar a mudanças na distribuição dos dados. Assim, o aprendizado online pode oferecer uma abordagem mais eficiente e eficaz, alinhada às características de cenários realistas para classificação de malware. Logo, este trabalho estabelece as seguintes perguntas de pesquisa:

P1: Qual das abordagens: aprendizado em lotes ou aprendizado incremental é mais adequada para lidar com classificação de malware ao longo do tempo ?

P2: Como a ocorrência *concept drift* se manifesta em um *dataset* de malware bancário real e sem seleção de famílias?

Para respondê-las, foram coletados 4.621 exemplares de malware advindos de *phishing* recebidos entre 2017 e 2022 (porém vistos pela primeira vez na Internet desde 2012). Esses exemplares foram posteriormente analisados utilizando uma ferramenta de análise dinâmica criada para monitorar a execução de programas em ambiente controlado [6]. As chamadas de cada programa executado são escritas em *logs* que compreendem os seguintes subsistemas do sistema operacional Windows: sistema de arquivos, processo e Registro. A rotulação dos *logs* foi feita com base na identificação da amostra gerada pelo antivírus *Microsoft Defender*, obtida via API do *VirusTotal* [7]. Aos dados coletados aplicou-se técnicas de processamento de linguagem natural a fim de analisar os *logs* gerados

com algoritmos de classificação. Foram feitos testes utilizando o *Term Frequency-Inverse Document Frequency* (TF-IDF) para extração de características e posterior classificação usando Árvores de Hoeffding. As principais contribuições deste trabalho são:

(1) Análise da presença de *concept drift* em conjunto de exemplares de malware especificamente voltados para Internet Banking brasileiro, coletados em cenário realista e modelados como fluxos de dados, cujas características foram extraídas por análise dinâmica;
(2) Discussão de diferentes abordagens da aplicação de técnicas de aprendizado de máquina para classificação de malware e seu impacto ao longo do tempo;
(3) Disponibilização dos códigos e *datasets* em https://github.com/secretrdlab/CotB-MalFi-Dataset.

2 Fundamentação Teórica

A seção atual tem por objetivo explicar brevemente os conceitos fundamentais por trás das técnicas utilizadas no presente trabalho, levando em consideração os dados brutos e sua coleta, a extração de características e a detecção de *concept drift*.

2.1 Defesa contra Códigos Maliciosos

Códigos maliciosos, ou malware, são programas (ou *scripts*) cuja intenção é violar as propriedades de segurança de um sistema, dispositivo e/ou usuário para fins que vão desde o comprometimento do alvo para uso ilícito até a exfiltração de informações sensíveis e ganhos financeiros. Anteriormente conhecidos por vírus de computador, esses programas têm como principal obstáculo os programas antivírus. Os primeiros programas maliciosos a se disseminarem na Internet como a conhecemos hoje (a partir de 1994) possuíam comportamentos predominantes que facilitavam sua classificação. Por exemplo, os vírus eram programas cuja atuação baseava-se em infectar arquivos pré-existentes, anexando-se a esses de forma que fossem executados junto da execução do programa infectado; os *worms*, por sua vez, espalhavam-se de maneira autônoma por uma rede em busca de determinados serviços vulneráveis que, ao serem encontrados, sofriam tentativa de exploração por meio da descarga do código infeccioso embutido na carga útil do *worm*; os cavalos de Tróia, ou *Trojans*, disfarçavam-se de programas legítimos (ou eram versões modificadas desses programas) com o objetivo de ludibriar os usuários a os executarem e, assim, comprometer o sistema e coletar dados sensíveis [8], entre outras classes.

Dentro desse contexto, os antivírus convencionais, que comumente atuam por meio da detecção baseada em assinaturas e heurísticas, tinham papel extremamente relevante como primeira linha de defesa. O funcionamento dos antivírus convencionais pode ser sumarizado da seguinte forma: enquanto a detecção baseada em assinaturas identifica padrões suspeitos via comparação de *strings* ou assinaturas conhecidas em arquivos e códigos (por exemplo, usando *fuzzy hashing* e regras Yara), a detecção baseada em heurísticas pode monitorar a execução do código em *sandbox* para mapear padrões de comportamento malicioso [9]. Além disso, antivírus comerciais costumam ter múltiplos "motores" para detecção de malware (incluindo modelos de aprendizado de máquina), cada um com suas limitações relacionadas à códigos evasivos ou análise de arquivos em formatos menos populares do que binários executáveis [10].

Do final da primeira década dos anos 2000 em diante, houve uma sofisticação maior por parte dos desenvolvedores de programas maliciosos que culminou na quase irrelevância da classificação por comportamentos predominantes únicos, dado que diversas situações começaram a ficar comuns: *worms* que carregavam vírus como carga útil [11]; malware multilinguagem e multi-componente para atacar sistemas ciberfísicos (ex.: Stuxnet [12]); a disseminação massiva de *malware* bancário em kits, especialmente no Brasil [13][14]; o surgimento e proliferação de *ransomware* [15] e, mais recentemente, malware com atuação multi-estágio [16]. Com isso, criou-se a necessidade de se aplicar técnicas mais complexas para a análise de malware que possam fomentar a definição de novas assinaturas/heurísticas. Para tanto, faz-se uso de sistemas de análise de malware cujas técnicas de monitoração podem ser categorizadas em análise estática e dinâmica.

Na análise estática, observa-se o binário suspeito sem que este entre em execução, seja pela análise de seu código-fonte (quando disponível) ou pela descompilação/*disassembling* e extração de *strings* e/ou chamadas de API [17]. Já a análise dinâmica envolve a execução do binário suspeito em um ambiente controlado, também chamado de *sandbox*, que pode ser um sistema real, virtual ou emulado no qual as ações do executável monitorado são registradas (por exemplo, modificações nos subsistemas e *downloads* efetuados, criação de novos arquivos e processos, interação com outras máquinas via rede) [18]. Os dados obtidos da análise estática e dinâmica podem também ser vistos como atributos que, ao serem processados, servem como características para representação desses programas como amostras de entrada de treinamento em algoritmos de aprendizado de máquina, visando a criação de modelos de detecção ou classificação [19]. A aplicação de técnicas de aprendizado de máquina agrega ao rol de soluções de proteção, mas também traz seus próprios problemas inerentes à area, tais como a ocorrência de *concept drifts* (discutido na Seção 2.3). Neste trabalho, utilizou-se de resultados provenientes da análise dinâmica em *sandbox* dos exemplares de malware, com rótulo atribuído por antivírus.

2.2 *Term Frequency-Inverse Document Frequency*

A TF-IDF é uma técnica utilizada para a recuperação de informações e processamento de linguagem natural (PLN) [20], com a finalidade principal de avaliar a relevância de palavras presentes em um dado documento com relação a um conjunto de documentos. Dessa forma, palavras mais comuns possuem uma pontuação mais baixa e palavras mais raras possuem uma pontuação mais alta, permitindo a remoção de termos desinteressantes ou que não permitam a distinção dos documentos e, consequentemente, comparação e classificação destes. Esse esquema de pontuações auxilia na transformação de texto em vetores de características que podem ser utilizados posteriormente por algoritmos de aprendizado de máquina para aprender contexto e representar dados baseados em atributos textuais, como é o caso dos dados obtidos de análise estática ou dinâmica de malware. Para o cálculo da TF-IDF, é necessário o cálculo da frequência do termo proposto (*Term Frequency* – TF) e da frequência inversa do documento (*Inverse Document Frequency* – IDF), dados pelas Equações 1 e 2, respectivamente.

$$\mathrm{TF}(t,d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}} \tag{1}$$

$$\text{IDF}(t, D) = \log\left(\frac{|D|}{1 + |\{d \in D : t \in d\}|}\right) \quad (2)$$

Na Equação 1, é expressa a frequência de um termo t em um documento d. Esse termo é uma palavra qualquer: nos *logs* de um *malware* pode ser uma chamada de API ou parâmetros dessa função. Nesse cenário, $f_{t,d}$ corresponde ao número de ocorrências do termo t no documento d enquanto $\sum_{t' \in d} f_{t',d}$ a soma das ocorrências de todos os termos no mesmo documento. Por sua vez, na Equação 2 é feito o calculo da frequência inversa do documento. O IDF é definido como o logaritmo da razão entre o número de total de documentos $|D|$ e o número de documentos em que o termo t aparece, dado por $|\{d \in D : t \in d\}|$. Os valores de TF e IDF são combinados para se obter o valor de TF-IDF, como disposto na Equação 3:

$$\text{TF-IDF}(t, d, D) = \text{TF}(t, d) \cdot \text{IDF}(t, D) \quad (3)$$

Conforme a Equação 3, a frequência de cada termo é atenuada pelo valor de IDF correspondente. O IDF atua como um fator de ajuste para aumentar a relevância de palavras que ocorrem com menor frequência no *corpus* e diminuir a relevância de termos mais comuns. Essa abordagem tem como objetivo destacar termos com maior capacidade discriminativa, contribuindo para uma melhor representação de palavras e documentos. Neste trabalho, optou-se por utilizar a TF-IDF para extração de características de exemplares de *malware*, usando chamadas de API como entrada. Isto deu-se devido a simplicidade do algoritmo e sua capacidade de extrair a relevância de termos com base em sua frequência em um documento e no corpus, possibilitando o encontro de padrões discriminantes associados ao comportamento de famílias de *malware*.

2.3 Concept Drift

Uma das maiores suposições ao lidar com aprendizado de dados é que um determinado *dataset* é gerado a partir de uma única função desconhecida e estática. Dessa forma, o papel de um modelo seria aprender essa função desconhecida a partir dos dados [21]. Entretanto, ao se lidar com fluxos contínuos de dados, a função geradora de um momento t não é a mesma em um instante t+1. Uma variação na função geradora é conhecida como *concept drift* [22], indicando que a mudança dos dados de entrada de um modelo causou a obsolescência do mesmo. Uma tarefa de classificação em aprendizado de máquina pode ser definida pelas probabilidades *a priori* das classes *P(y)* e a função de densidade de probabilidade condicional *P(X|y)* [23]. Logo, uma classificação pode ser definida pela Teoria de Decisão Bayesiana, mostrada na Equação 4.

$$P(y|X) = \frac{P(y)P(X|y)}{P(X)} \quad (4)$$

Com base na Equação 4, um *concept drift* pode ser definido quando há uma mudança nas probabilidades *a posteriori* ao longo do tempo (Equação 5) [22, 24, 25].

$$P_t(y|X) \neq P_{t+1}(y|X) \quad (5)$$

onde t é um instante de tempo. O *concept drift* definido na Equação 5 é comumente referido como um *concept drift* real, onde a relação entre os vetores de características e as classes alvo muda com o tempo, modificando mesmo a fronteira de decisão ideal entre os tempos t e $t + 1$. Mudanças nas probabilidades *a priori*, definidas por $P_t(y) \neq P_{t+1}(y)$, ou na distribuição dos dados, definidas por $P_t(X) \neq P_{t+1}(X)$, são comumente chamadas de *concept drifts* virtuais. Outras definições e discussões mais aprofundadas sobre os tipos de concept drifts podem ser vistas em [5, 26].

Tanto *concept drifts* reais quanto virtuais podem levar a deterioração da acurácia de classificadores ao longo do tempo [5], o que é ainda mais grave em dados que podem sofrer mutações frequentes, como é o caso dos malwares. Dentre as técnicas disponíveis para detecção de *concept drifts* em fluxos de dados, uma das mais populares é a utilização de gatilhos que monitoram as distribuições dos dados e sinalizam possíveis mudanças. Um desses métodos é o ADWIN—*ADaptive WINdowing* [27]—que avalia mudanças súbitas na média e na variância de janelas deslizantes adaptativas em relação aos dados observados. Neste trabalho, usa-se o ADWIN como gatilho de detecção de *concept drifts* no fluxo de dados de exemplares de malware ao longo do período observado nos experimentos.

3 Árvore de *Hoeffding*

Árvores de Hoeffding, também chamadas de árvore de decisão super rápidas, é um algoritmo de aprendizado de máquina baseado em árvores de decisão [28]. Uma árvore de decisão aprende de maneira recursiva dividindo os dados em *subsets* baseados em valores de atributos. Durante o processo de treinamento todos os atributos são comparados entre si utilizando uma heurística afim de determinar qual o melhor atributo deve ser usado para dividir os dados. No caso de árvores de Hoeffding é utilizado o limiar de Hoeffding para determinar qual atributo será escolhido, o qual quantifica matematicamente o número de observações necessárias para determinar se um atributo será uma boa escolha [29].

4 Trabalhos Relacionados

Vários métodos foram propostos para lidar com *concept drifts* no âmbito da detecção de malware. Jordaney et al. [30] e Kumpulainen et al. [31], por exemplo, propõem novas abordagens para a implementação de gatilhos considerando conjuntos de dados construídos a partir da análise estática de artefatos maliciosos capturados em sistemas operacionais *MS Windows*. Barbero et al. [32] estende o *framework* estatístico Transcend [30] de forma que este seja formalizado e se torne uma plataforma aplicável na prática. Já Hu et al. [33] e Guerra-Manzanares et al. [34] também propõem novos gatilhos para detecção de *concept drifts*, porém para amostras de malware cujo alvo é o sistema operacional *Android*.

Tais abordagens lidam com o problema da detecção de *concept drift* em classificadores/detectores de malware considerando lotes de amostras (*batch*), diferente do presente trabalho que foca em analisar classificadores utilizando aprendizado incremental em um fluxo continuo de dados. A separação das amostras por lotes pode implicar em diversos problemas, como a necessidade de aguardar até que o lote a ser analisado acumule amostras suficientes antes que o próximo lote seja enviado para o detector, ou o desbalanceamento severo de amostras em algumas classes pode comprometer os resultados apresentados por um dado modelo [35]. Além disso, a natureza sazonal e adaptativa das campanhas de infecção por famílias de malware e a própria dependência temporal da ocorrência de *concept drift* parecem ser melhor tratadas se considerarmos modelar

o problema como um problema de detecção de *concept drifts* em fluxos de instâncias únicas (fluxo online) [4].

Um ponto a se considerar é que diversos estudos na literatura propõem a criação de novos conjuntos de dados de malware. Por exemplo, Yang et al. [36] apresentam os BODMAS, um *dataset* de classes de *malware* para MS Windows (arquivos do tipo "exe", ou *Portable Executable*) com amostras coletadas entre 2019 e 2020 e criado para abordar o problema de *concept drift*. Apesar de se tratar de um dos poucos *datasets* disponíveis que exploram *concept drift* no contexto de executáveis maliciosos, suas características foram extraídas unicamente por análise estática, diferente do presente trabalho. Outro conjunto de dados feito a partir de executáveis para MS Windows propõe também um *benchmark* baseado em chamadas de *API* para a classificação malware [37]. A coleta das chamadas de API das amostras em execução foi feita com a *sandbox Cukoo* [38] e os rótulos foram obtidos usando o *VirusTotal*. Ao todo o conjunto de dados contém 7.101 exemplares coletados durante quatro meses de 2018 a partir de repositórios abertos no *GitHub*. Embora as características tenham sido extraídas via análise dinâmica, tal qual o presente trabalho, a ferramenta *Cuckoo* tem várias limitações e parou de ser atualizada em 2016 [39].

Jiang et al. [40] disponibilizou um conjunto de malware rotulado com auxílio do *VirusTotal*, contendo 223 mil amostras de 526 famílias diferentes com foco em classificar exemplares em famílias sob a presença de *concept drift*. As características fornecidas no conjunto, entretanto, são extraídas com análise estática, sendo comparável a *benchmarks*/conjuntos como o *EMBER* [41] e Byte [42], ambos compostos por amostras previamente classificadas, baseados em aprendizado estático e sem ênfase na evolução temporal. Em Ceschin et al. [4], por sua vez, avaliam o impacto de *concept drift* durante a classificação de malware para dois conjuntos de dados disponíveis na literatura baseados no sistema *Android*: DREBIN [43] e um subconjunto do AndroZoo [44]. De modo similar ao presente trabalho, utilizou-se em [4] TF-IDF para extração de características, classificadores baseados em árvores de decisão e gatilhos para detecção de *concept drift*, porém as características dos *datasets* vêm da análise estática e os exemplares são de *Android*.

Muitos trabalhos usam dados advindos da análise estática dos exemplares para conduzir experimentos. Porém, este trabalho propõe uma nova abordagem, realizando análises com base em dados de análise dinâmica em uma *stream*. Para isso, usou-se chamadas de API obtidas por meio de análise dinâmica para capturar comportamentos que de fato foram executados pelos exemplares, oferecendo uma representação mais fiel das ameaças. Na análise dos resultados, discute-se principalmente as diferenças de métodos de aprendizado baseado em lote e aprendizado incremental para classificação de malware em cenários dinâmicos e sujeitos a *concept drift*.

5 Metodologia e Experimentos

Nesta Seção são descritos os procedimentos adotados para a condução deste trabalho, abrangendo desde a coleta e seleção dos dados até a definição e execução dos experimentos, passando pela preparação e extração de informações relevantes para análise.

5.1 Conjunto de Dados

Nesta seção, descrevemos o conjunto de dados utilizado para investigar a presença de *concept drift* em *malwares* bancários distribuídos por meio de campanhas de *phishing* via e-mail institucional do laboratório. No total, foram coletados 4.621 exemplares únicos com datas de aparição entre 2012 e 2022 (com base no atributo *first_seen_date* obtido do relatório da amostra no VirusTotal).

Os 4.621 exemplares de malware do conjunto inicial estão distribuídos em 202 famílias distintas que, para fins de consistência e facilidade de rotulação, usou-se o resultado do antivírus Microsoft Defender obtido a partir do relatório da amostra provido pelo VirusTotal. A justificativa para essa escolha é que cada antivírus possui motor e esquema de nomeação próprios, causando inconsistência nos resultados mesmo que se utilize de ferramentas de *tagging* automáticas para votação de rótulos como AVClass [45]. Além disso, as regras de proteção/assinaturas do Microsoft Defender são construídas sobre uma base de dados que compreende a maior parte dos usuários de desktops e é nativo e habilitado por padrão nos sistemas MS Windows atuais.

Durante a análise dos dados, foi realizado um pré-processamento que envolveu a seleção das famílias com mais de 15 amostras. Após a aplicação desse critério, restaram 1.356 amostras, distribuídas entre 15 classes distintas, conforme detalhado na Tabela 1. Esse conjunto foi utilizado para os experimentos conduzidos neste trabalho.

Tabela 1: Relação dos nomes das famílias de malware consideradas nos experimentos e total de exemplares por família. Foram selecionadas apenas aquelas que possuíam mais do que 15 representantes, reduzindo o conjunto inicial para 1.356 exemplares únicos de malware bancário.

Família	# Exemplares
Banload	421
Banker	182
Ymacco	108
Tiggre	104
Wacatac	95
Dynamer	73
AgentTesla	71
Occamy	54
Skeeyah	52
Bladabindi	41
SodinokibiCrypt	34
Valla	32
Fareit	29
FormBook	21
Bancos	20
Sabsik	19
Total	**1.356**

Ao lidar com dados de segurança relacionados a malware, é fundamental considerar dois comportamentos característicos frequentemente observados no fluxo de dados: (1) as variações entre exemplares de uma mesma família [46]; e (2) o comportamento populacional das famílias de *malware*, que geralmente exibem um período inicial de maior prevalência, seguido por uma tendência

de diminuição ao longo do tempo [47]. Esses comportamentos são intrínsecos a esse tipo de fluxo de dados, em grande parte devido a evolução dos códigos maliciosos, que visam contornar as atualizações dos mecanismos de defesa. A Figura 1 ilustra a distribuição de cada família ao longo do fluxo de dados no período de coleta dos exemplares, evidenciando tais variações e tendências temporais.

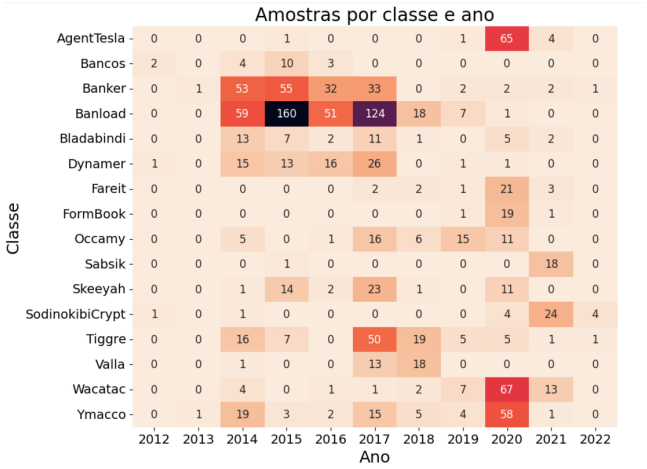

Figura 1: População de famílias ao longo da *stream* de dados, considerando-se a data de primeira aparição no VirusTotal.

Pode-se observar que ao longo do fluxo, os exemplares concentram-se entre os anos de 2014 e 2021. As famílias *Banload*, *Banker* e *Dynamer* apresentam grande prevalência durante o período de 2014 a 2018, com uma redução considerável em sua captura a partir desse ponto. O período subsequente, de 2018 a 2022, é dominado por outras famílias, como *Farei*, `FormBook`, *Wacatac*, *AgentTesla* e *Ymacco*. Esse padrão de ascensão e declínio das famílias de malware reflete a dinâmica de adaptação entre os sistemas de defesa e as estratégias dos atacantes, que partem para outras famílias ou mudam sensivelmente as variantes até que uma família se desdobre em outra para evadir a detecção desses exemplares de malware.

Esse comportamento observado na população em determinado período de tempo deve ser considerado ao se escolher a abordagem mais apropriada para o desenvolvimento de modelos de aprendizado de máquina, uma vez que a evolução temporal das ameaças pode influenciar diretamente na eficácia desses modelos. Um modelo de aprendizado de máquina treinado em dados antigos pode não ser eficaz em detectar variantes ou ameaças emergentes, tornando-se obsoleto. Assim, a criação de modelos robustos exige atualizações frequentes e uma abordagem flexível capaz de capturar padrões e tendências que se modificam com o tempo, isto é, estar preparado para a presença de *concept drifts*.

5.2 Processamento das Amostras

Neste trabalho, cada exemplar de *malware* coletado passa por um fluxo de processamento responsável por transformar um binário executável em um vetor numérico utilizado pelos classificadores para os testes propostos, seguindo as etapas do *pipeline* apresentado abaixo.

Execução de amostras. As amostras coletadas foram executados em ambiente controlado, utilizando ferramenta própria de análise dinâmica [6], que captura algumas ações no formato de *logs* textuais. Durante o processo de análise dinâmica, foram monitoradas atividades nos subsistemas de arquivos, processos e registro do sistema operacional MS Windows 8. Ao final da análise é gerado um arquivo-texto de *log* para cada subsistema monitorado. Para se obter uma visão unificada da execução, os três arquivos de *logs* obtidos foram mesclados em um único arquivo textual, considerando a ordem de escrita baseada no tempo do momento da captura da operação.

Formato dos Logs. O *log* de processo possui as colunas: tempo do momento da captura, operação realizada (leitura, escrita, criação, término, remoção), o PID - identificador de processo do executável analisado (possivelmente um malware), o caminho do executável analisado (processo responsável pela operação feita) e o caminho do processo impactado pela ação (alvo). Os *logs* do subsistema de registro e arquivos possuem uma estrutura semelhante, com a adição de colunas extras como o valor da escrita para o *log* de arquivo (em geral o nome do arquivo alvo da operação) e o valor da chave do registro.

Extração de Características. O algoritmo escolhido para a extração foi o TF-IDF utilizando os parâmetros padrão da biblioteca `scikit-learn`, com exceção do argumento *stops_words*, que foi configurado para considerar somente os caracteres '|' (caracter separador) e '\n'. O tamanho do vetor de características foi definido como 64 (cada coluna do vetor representa o valor de TF-IDF calculado para uma das 64 palavras selecionadas pelo algoritmo como mais relevantes para representação dos exemplares). Para os experimentos, adotou-se como padrão o uso dos primeiros 25% dos dados para o treinamento do TF-IDF, dessa forma, o restante dos dados foram apenas transformados com o TF-IDF treinado. Essa divisão de treino e teste foi feita para garantir que não houvesse vazamento de dados/características futuras e evitar problemas de aplicação incorreta de aprendizado de máquina em dados de segurança [35].

5.3 Definição dos Experimentos

Para avaliar a abordagem mais adequada para lidar com *logs* de execução de programas maliciosos e validar a hipótese de considerar malwares como um fluxo de dados, foram consideradas duas estratégias amplamente utilizadas em aprendizado de máquina: o aprendizado estático, também chamado de aprendizado em lotes, e o aprendizado incremental, também chamado de aprendizado *online*. No aprendizado estático, os dados são particionados em dois subconjuntos fixos, destinados ao treinamento e à avaliação do modelo, respectivamente. Essa abordagem pressupõe que a distribuição dos dados permanece estável ao longo do tempo [48].

Em contraste, o aprendizado incremental lida com fluxos contínuos de dados (*data streams*), permitindo que o modelo se adapte de forma imediata a novas instâncias, sem a necessidade de acumular dados para um novo treinamento [21]. Dessa forma, o modelo é capaz de se adaptar às mudanças no fluxo de dados.

Para a realização dos experimentos, foi utilizado o classificador *Hoeffding Tree*, também conhecido como árvore de decisão extremamente rápida (*Very Fast Decision Tree*) [29]. Esse algoritmo foi escolhido devido à sua capacidade de aprendizado *online*, o que o

torna adequado para lidar com fluxos contínuos de dados em cenários de grandes volumes. Em todos os experimentos foi utilizada a implementação da biblioteca `capymoa`, versão 0.7.0, desenvolvida em *Python*. Os parâmetros do classificador foram mantidos em seus valores padrão, com exceção do *grace_period*, que determina o número mínimo de instâncias que um nó folha necessita antes de criar um novo nó de teste, que foi definido como 15. Esse valor torna a criação de novos nós mais provável, o que é necessário já que o conjunto de dados utilizado possui pouco mais de 1.300 amostras.

Métricas de Avaliação. A métrica selecionada para avaliar os experimentos foi a acurácia definida na Equação 6. Na qual, n representa o número total de amostras, y_i corresponde ao rótulo verdadeiro da i-ésima amostra e $\hat{y}_i$ indica o rótulo predito para a i-ésima amostra. Nos experimentos de aprendizado estático, a acurácia foi calculada com base no desempenho no conjunto de dados de treino e teste. Já para os experimentos de aprendizado incremental, foi utilizada a acurácia prequencial. Esta é determinada pela Equação 6, porém aplicada ao longo de uma janela deslizante de tamanho fixo (32, neste caso) e de deslocamento igual a um.

$$\text{Acurácia} = \frac{\sum_{i=1}^{n}(y_i = \hat{y}_i)}{n} \tag{6}$$

Aprendizado Estático. Para testar o paradigma de treinamento em lotes, foi feito o treinamento de árvore de Hoeffding sem gatilhos de detecção de *concept drift* e considerando diferentes configurações de conjuntos de dados: no primeiro experimento, os 25% iniciais do fluxo de dados foram utilizados para treinamento, enquanto os 75% restantes foram destinados ao teste (denominado *Dataset 1*); no segundo, 50% dos dados iniciais foram usados para treinamento e os 50% restantes para teste (*Dataset 2*); por fim, no terceiro experimento, 75% iniciais dos dados foram empregados para treinamento, enquanto os 25% finais foram utilizados para teste (*Dataset 3*).

Aprendizado Incremental. Outro paradigma avaliado foi o aprendizado incremental, no qual o protocolo a seguir foi adotado: comparar a acurácia prequencial de quatro árvores de Hoeffding, cada uma com uma estratégia distinta de treinamento, seguindo a abordagem proposta em [24]. O primeiro modelo, denominado estático, foi treinado utilizando 25% da porção inicial do fluxo de dados. O segundo modelo foi treinado ao longo do fluxo de dados com a estratégia de teste-treino, na qual o modelo é primeiramente testado com uma amostra e, em seguida, treinado com a mesma (utilizando seu rótulo real). O terceiro modelo foi atualizado continuamente assim como o teste-treino, porém teve a adição de um gatilho de *concept drift ADWIN*. Quando um *concept drift* é detectado pelo gatilho, o modelo é descartado e substituído por novo, treinado com as últimas 100 amostras observadas. O quarto modelo é descartado e atualizado continuamente, sendo treinado a partir de uma janela deslizante que contém as 339 amostras mais recentes do fluxo de dados (25% do fluxo).

6 Resultados e Discussão

Nesta Seção são apresentados os resultados obtidos com os experimentos definidos na Seção 5.3.

6.1 Resultados do Aprendizado em Lotes

A fim de reproduzir a abordagem mais comum de treinamento e teste em aprendizado de máquina, foram realizados experimentos variando a proporção de dados destinados ao treinamento e teste de uma árvore de Hoeffding. Ao analisar a Tabela 2, observa-se que a acurácia durante o treinamento variou entre 38,39% e 49,85% entre os diferentes *datasets*. Vale ressaltar que os os valores obtidos na tabela, embora pareçam baixos (probabilidade menor do que 50% é comumente associado com a falácia de que o resultado é pior do que o lançamento de uma moeda), são bem mais altos do que simplesmente chutar uma classe, uma vez que tem-se 15 famílias e a probabilidade de acerto p esperada é dada por 1/15 ($p < 7\%$).

Tabela 2: Acurácia calculada para abordagem de aprendizado estático. Os *Datasets* 1, 2 e 3 utilizam 25%, 50% e 75% dos dados para treinamento, respectivamente. O restante dos dados é utilizado para teste.

	Dataset 1	**Dataset 2**	**Dataset 3**
Treino	38,39%	49,85%	40,41%
Teste	27,63%	10,91%	0,08%

O maior valor de acurácia no treinamento foi alcançado utilizando o *Dataset 2*, onde metade dos dados foi utilizada para o treinamento e teste do modelo. Embora a acurácia na fase de treinamento tenha se mantido próxima de 45% em todos os cenários, houve uma queda significativa na acurácia durante a fase de teste em todas as configurações avaliadas. Em geral, ao utilizar aprendizado estático com modelos de aprendizado de máquina, espera-se que o desempenho do modelo melhore à medida que mais dados são disponibilizados para o treinamento. Contudo, essa tendência não foi observada nos testes. O treinamento realizado com o *Dataset 3*, que possuía a maior quantidade de dados, apresentou o pior desempenho na fase de teste, com acurácia inferior a 1%. Esse comportamento sugere a presença de *concept drifts* nos dados, uma vez que os modelos tendem a perder acurácia ao longo do tempo e a introdução de mais dados rotulados sem o descarte de informações conflitantes afeta negativamente os resultados. Assim, os dados utilizados no treinamento podem não representar adequadamente os dados futuros utilizados no teste, o que é corroborado pelas mudanças na população das famílias durante a *stream*.

6.2 Resultados do Aprendizado Incremental

Para avaliar o aprendizado incremental, foram treinadas quatro Árvores de Hoeffding com os mesmos parâmetros definidos anteriormente, porém com diferentes estratégias de aprendizado. Durante o processamento do fluxo de dados, mediu-se a acurácia prequencial dos modelos, cujos resultados são mostrados na Figura 2.

Por terem sido treinados com o mesmo conjunto de dados (primeiros 20% do fluxo), os modelos apresentam desempenho similar no início do fluxo, diferenciando-se progressivamente ao longo do teste. O modelo estático demonstra acurácia semelhante ao modelo *test-then-train* durante grande parte das instâncias de teste. Entretanto, próximo à instância 600, observa-se uma queda acentuada em sua acurácia. Essa redução abrupta é mitigada nos demais modelos, que são continuamente atualizados com a incorporação de novos

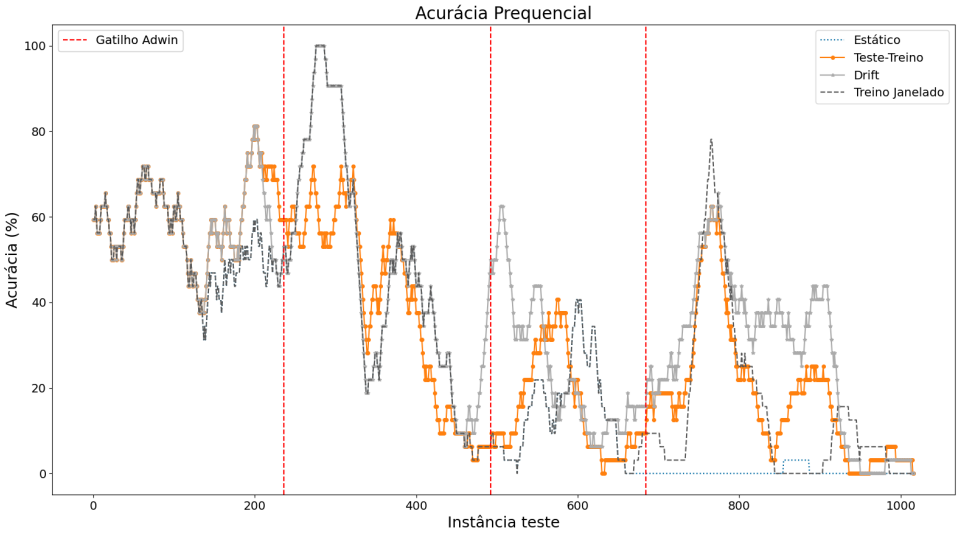

Figura 2: Acurácia prequencial considerando uma janela de tamanho 32. As linhas verticais tracejadas indicam os pontos em que o gatinho sinalizou a presença de um *concept drift*.

dados. Esse resultado é corroborado pela Tabela 3, onde é exibida a acurácia final obtida para cada uma das abordagens.

A estratégia Teste-Treino demonstrou uma acurácia superior ao do modelo estático, indicando que modelos treinados no passado podem não ter recebido uma amostragem de dados suficientemente representativas do problema. Como o modelo Teste-Treino somente agrega novos dados, sem esquecer dados passados, isso é um indicativo da presença de um possível *concept drift* virtual, onde algumas regiões do espaço do problema podem ser desconhecidas no início do fluxo [23, 26].

A estratégia do modelo *Drift* alcançou a maior acurácia ao longo de todo o fluxo. Os pontos em que o gatilho sinalizou a presença de *concept drifts* estão destacados como linhas tracejadas verticais vermelhas na Figura 2. Esses resultados indicam que a remoção de dados do passado pode ser benéfica. Isso pode indicar a presença de *concept drifts* reais, onde as probabilidades *a posteriori* podem ser alteradas com o tempo, e a simples adição de dados nos conjuntos de treino dos classificadores pode não ser suficiente caso dados passados conflitantes não sejam removidos [23, 26].

Já a abordagem janelada mostrou uma acurácia maior que o modelo estático, e em alguns momentos superior à todas as demais abordagens considerando o gráfico da Figura 2. No entando, essa abordagem levou a uma acurácia final inferior à abordagem que esquece os dados apenas quando um *concept drift* é detectado. Isso indica que esquecer os dados passados é importante para esse problema, no entanto, promover esse esquecimento somente em momentos em que o comportamento dos dados muda é mais eficaz.

7 Conclusão

Nesse trabalho, foram conduzidos experimentos com diferentes paradigmas de treinamento de modelos de aprendizado de máquina a fim de determinar uma forma adequada para lidar com a classificação de malware ao longo do tempo, trantando tais amostras como

Tabela 3: Acurácia média e desvio padrão para testes feitos com aprendizado incremental.

Abordagem	Acurácia	Desvio Padrão
Estático	26,8%	±27, 8
Teste-Treino	32,7%	±23, 6
Drift	39,2%	±23, 4
Treino Janelado	31,7%	±26, 0

um fluxo de dados contínuo, em vez de um lote de dados como é comum na literatura. Para validar as hipóteses feitas e responder às perguntas definidas neste trabalho, foram realizados testes considerando exemplares de malware bancário coletados entre os anos de 2017 e 2022, cujos resultados fomentaram as seguintes conclusões:
P1: Qual das abordagens: aprendizado em lotes ou aprendizado incremental é mais adequada para lidar com classificação de malware ao longo do tempo ? Ao utilizar aprendizado incremental, os experimentos mostraram resultados superiores aos testes feitos com abordagem de aprendizado estático. Ao longo do fluxo de dados os classificadores perdem acurácia abruptamente principalmente na porção final do conjunto de dados. Essa queda de performance foi menor ao utilizar o aprendizado incremental devido a atualização e adaptação contínua dos classificadores.
P2: Como a ocorrência *concept drift* se manifesta em um conjunto de dados de malware bancário real e sem seleção de famílias? Ao longo dos testes a abordagem de maior desempenho foi a que contava com um gatilho de *concept drift*. Em determinados pontos do fluxo de dados foi mais vantajoso para o modelo esquecer um conhecimento aprendido com dados mais antigos do que rete-los, pois esses já não eram representates do momento atual. Esse comportamento observado indica a presença de *concept drift* no fluxo de dados coletados [24]. Essa evolução dos malwares se

mostrou ser devido a sazonalidade de campanhas, já que as classes no fim do fluxo de dados eram diferentes das observadas no começo.

8 Agradecimentos

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

Referências

[1] Gilberto Costa. Golpes bancários se espalham e destroem vida financeira de vítimas. https://agenciabrasil.ebc.com.br/economia/noticia/2024-01/golpes-bancarios-se-espalham-e-destroem-vida-financeira-de-vitimas, 2024.

[2] Vitoria Lopes Gomez. Golpe: Brasil é o segundo país com mais crimes digitais no mundo. https://olhardigital.com.br/2023/10/27/seguranca/golpe-brasil-e-o-segundo-pais-com-mais-crimes-digitais-no-mundo/, 2023.

[3] Kaspersky Team. Ataques contra celular crescem 70% e atingem número histórico na américa latina. https://www.kaspersky.com.br/blog/panorama-ameacas-latam-2024/, 2024.

[4] Fabrício Ceschin, Marcus Botacin, Heitor Murilo Gomes, Felipe Pinagé, Luiz S Oliveira, and André Grégio. Fast furious: On the modelling of malware detection as an evolving data stream. *Expert Systems with Applications*, 212, 2023.

[5] Jie Lu, Anjin Liu, Fan Dong, Feng Gu, Joao Gama, and Guangquan Zhang. Learning under concept drift: A review. *IEEE Transactions on Knowledge and Data Engineering*, page 1–1, 2018. ISSN 2326-3865. doi: 10.1109/tkde.2018.2876857. URL http://dx.doi.org/10.1109/TKDE.2018.2876857.

[6] Marcus Botacin, Fabrício Ceschin, and André Grégio. Corvus: Uma solução sandbox e de threat intelligence para identificação e análise de malware. In *Anais Estendidos do XXI SBSeg*. SBC, 2021.

[7] VirusTotal. Virustotal - free online virus, malware and url scanner. https://www.virustotal.com/, 2024.

[8] Eric Filiol. *Computer Viruses: From Theory to Applications*. Springer, 2005.

[9] Efstratios Chatzoglou, Georgios Karopoulos, Georgios Kambourakis, and Zisis Tsiatsikas. Bypassing antivirus detection: old-school malware, new tricks. In *Proceedings of the 18th International Conference on Availability, Reliability and Security*, ARES '23, 2023.

[10] Antiviruses under the microscope: A hands-on perspective. *Computers & Security*, 112:102500, 2022.

[11] Thomas M Chen and Jean-Marc Robert. The evolution of viruses and worms. In *Statistical methods in computer security*, pages 289–310. CRC press, 2004.

[12] Thomas M Chen and Saeed Abu-Nimeh. Lessons from stuxnet. *Computer*, 44(4): 91–93, 2011.

[13] Lorenzo Martignoni, Roberto Paleari, and Danilo Bruschi. A framework for behavior-based malware analysis in the cloud. In *Information Systems Security: 5th International Conference, ICISS*, 2009.

[14] Fernando Mercês. The brazilian underground market. https://www.trendmicro.de/cloud-content/us/pdfs/security-intelligence/white-papers/wp-the-brazilian-underground-market, 2014.

[15] Harun Oz, Ahmet Aris, Albert Levi, and A Selcuk Uluagac. A survey on ransomware: Evolution, taxonomy, and defense solutions. *ACM Computing Surveys (CSUR)*, 54, 2022.

[16] Marcus Carpenter and Chunbo Luo. Behavioural reports of multi-stage malware. *arXiv preprint arXiv:2301.12800*, 2023.

[17] Rami Sihwail, Khairuddin Bin Omar, and Khairul Akram Zainol Ariffin. A survey on malware analysis techniques: Static, dynamic, hybrid and memory analysis. *International Journal on Advanced Science, Engineering and Information Technology*, 2018. URL https://api.semanticscholar.org/CorpusID:54017419.

[18] Vasilis Vouvoutsis, Fran Casino, and Constantinos Patsakis. Beyond the sandbox: Leveraging symbolic execution for evasive malware classification. *Computers & Security*, 149, 2025.

[19] Rajchada Chanajitt, Bernhard Pfahringer, and Heitor Murilo Gomes. Combining static and dynamic analysis to improve machine learning-based malware classification. In *2021 IEEE 8th International Conference on Data Science and Advanced Analytics (DSAA)*, 2021.

[20] Claude Sammut and Geoffrey I. Webb. Tf–idf. In *Encyclopedia of Machine Learning*. 2010. URL https://doi.org/10.1007/978-0-387-30164-8_832.

[21] Steven C.H. Hoi, Doyen Sahoo, Jing Lu, and Peilin Zhao. Online learning: A comprehensive survey. *Neurocomputing*, 459, 2021.

[22] T. Ryan Hoens, Robi Polikar, and Nitesh V. Chawla. Learning from streaming data with concept drift and imbalance: an overview. *Progress in Artificial Intelligence*, 2012.

[23] João Gama, Indrundefined Žliobaitundefined, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. A survey on concept drift adaptation. *ACM Comput. Surv.*, 46(4), March 2014. ISSN 0360-0300. doi: 10.1145/2523813. URL https://doi.org/10.1145/2523813.

[24] Paulo R Lisboa de Almeida, Luiz S Oliveira, Alceu de Souza Britto, and Jean Paul Barddal. Naïve approaches to deal with concept drifts. In *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 1052–1059. IEEE, 2020.

[25] Geoffrey I. Webb, Roy Hyde, Hong Cao, Hai Long Nguyen, and Francois Petitjean. Characterizing concept drift. *Data Mining and Knowledge Discovery*, 30(4):964–994, 2016. ISSN 1573-756X. doi: 10.1007/s10618-015-0448-4. URL https://doi.org/10.1007/s10618-015-0448-4.

[26] Paulo RL Almeida, Luiz S Oliveira, Alceu S Britto Jr, and Robert Sabourin. Adapting dynamic classifier selection for concept drift. *Expert Systems with Applications*, 104:67–85, 2018.

[27] Albert Bifet and Ricard Gavalda. Learning from time-changing data with adaptive windowing. In *Proceedings of the 2007 SIAM international conference on data mining*, pages 443–448. SIAM, 2007.

[28] Geoff Hulten, Laurie Spencer, and Pedro Domingos. Mining time-changing data streams. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 97–106, 2001.

[29] Asmah Muallem, Sachin Shetty, Jan W. Pan, Juan Zhao, and Biswajit Biswal. Hoeffding tree algorithms for anomaly detection in streaming datasets: A survey. *Computational Modeling & Simulation Engineering Faculty Publications*, 2017. URL https://digitalcommons.odu.edu/msve_fac_pubs/17. Publication No. 17.

[30] Roberto Jordaney, Kumar Sharad, Santanu K. Dash, Zhi Wang, Davide Papini, Ilia Nouretdinov, and Lorenzo Cavallaro. Transcend: Detecting concept drift in malware classification models. In *26th USENIX Security Symposium (USENIX Security 17)*, 2017.

[31] Iiro Kumpulainen et al. Adapting to concept drift in malware detection. Master's thesis, Aalto University, 2020.

[32] Federico Barbero, Feargus Pendlebury, Fabio Pierazzi, and Lorenzo Cavallaro. Transcending TRANSCEND: revisiting malware classification in the presence of concept drift. In *43rd IEEE Symposium on Security and Privacy*, 2022.

[33] Donghui Hu, Zhongjin Ma, Xiaotian Zhang, Peipei Li, Dengpan Ye, and Baohong Ling. The concept drift problem in android malware detection and its solution. *Security and Communication Networks*, 2017, 2017.

[34] Alejandro Guerra-Manzanares, Marcin Luckner, and Hayretdin Bahsi. Android malware concept drift using system calls: Detection, characterization and challenges. *Expert Systems with Applications*, 206, 2022.

[35] Daniel Arp, Erwin Quiring, Feargus Pendlebury, Alexander Warnecke, Fabio Pierazzi, Christian Wressnegger, Lorenzo Cavallaro, and Konrad Rieck. Dos and don'ts of machine learning in computer security. In *31st USENIX Security Symposium (USENIX Security 22)*, 2022.

[36] Limin Yang, Arridhana Ciptadi, Ihar Laziuk, Ali Ahmadzadeh, and Gang Wang. Bodmas: An open dataset for learning based temporal analysis of pe malware. In *2021 IEEE Security and Privacy Workshops (SPW)*, 2021.

[37] Ferhat Ozgur Catak and Ahmet Faruk Yazı. A benchmark api call dataset for windows pe malware classification. *arXiv preprint arXiv:1905.01999*, 2019.

[38] Cuckoo Sandbox Team. Cuckoo sandbox. https://www.cuckoosandbox.org. Accessed: 2024-11-17.

[39] Umoh Essien and Sylvester Ele. Cuckoo sandbox and process monitor (procmon) performance evaluation in large-scale malware detection and analysis. *British Journal of Computer Networking and Information Technology*, 7, 2024.

[40] Yongkang Jiang, Gaolei Li, Shenghong Li, and Ying Guo. Benchmfc: A benchmark dataset for trustworthy malware family classification under concept drift. *Computers & Security*, 139, 2024.

[41] Hyrum S Anderson and Phil Roth. Ember: an open dataset for training static pe malware machine learning models. *arXiv preprint arXiv:1804.04637*, 2018.

[42] Edward Raff, Richard Zak, Russell Cox, Jared Sylvester, Paul Yacci, Rebecca Ward, Anna Tracy, Mark McLean, and Charles Nicholas. An investigation of byte n-gram features for malware classification. *Journal of Computer Virology and Hacking Techniques*, 14, 2018.

[43] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, Konrad Rieck, and CERT Siemens. Drebin: Effective and explainable detection of android malware in your pocket. In *Ndss*, volume 14, pages 23–26, 2014.

[44] Kevin Allix, Tegawendé F Bissyandé, Jacques Klein, and Yves Le Traon. Androzoo: Collecting millions of android apps for the research community. In *Proceedings of the 13th international conference on mining software repositories*, pages 468–471, 2016.

[45] Marcos Sebastián, Richard Rivera, Platon Kotzias, and Juan Caballero. Avclass: A tool for massive malware labeling. In *Research in Attacks, Intrusions, and Defenses*. Springer International Publishing, 2016.

[46] Vinita Verma, Sunil K. Muttoo, and V.B. Singh. Multiclass malware classification via first- and second-order texture statistics. *Computers Security*, 97, 2020.

[47] Omar Alrawi, Charles Lever, Kevin Valakuzhy, Ryan Court, Kevin Snow, Fabian Monrose, and Manos Antonakakis. The circle of life: A Large-Scale study of the IoT malware lifecycle. In *30th USENIX Security Symposium (USENIX Security 21)*, 2021.

[48] Bartosz Krawczyk, Leandro L. Minku, João Gama, Jerzy Stefanowski, and Michał Woźniak. Ensemble learning for data stream analysis: A survey. *Information Fusion*, 37:132–156, 2017.