PEVuln: A Benchmark Dataset for Using Machine Learning to Detect Vulnerabilities in PE Malware

Nathan Ross[1,*], Oluwafemi Olukoya[1], Jesús Martínez del Rincón[1] and Domhnall Carlin[1]

[1]*Centre for Secure Information Technologies (CSIT), Queen's University Belfast*

Abstract

In this paper, we present a benchmark dataset for training and evaluating static PE malware machine learning models, specifically for detecting known vulnerabilities in malware. Our goal is to enable further research in defense against malware by exploiting their bugs or weaknesses. After recognising limitations in current malware datasets regarding exploitable malware, our dataset addresses these gaps by utilizing the malware vulnerability database *Malvuln*, and software vulnerability database *ExploitDB* to create a new malware dataset with 684 vulnerable malware samples, 35,241 non-vulnerable malware samples, 1,425 vulnerable benign samples, and 7,905 non-vulnerable benign samples, detailed with timestamps, families, threat mapping, vulnerability mapping, and obfuscation analysis. This 4-class dataset lays the foundation for advancing future research in analysis and vulnerability exploitation in malware using machine learning. We also provide baseline results using state-of-the-art models for malware classification to benchmark the performance of the dataset, where the binary tasks achieve F1 scores above 0.90, while the multi-class task attains an F1-Score of 0.958.

Keywords

Dataset, Machine Learning, Malware, Vulnerabilities

1. Introduction

Given the rapid evolution of cybersecurity and cyberattacks, relying solely on techniques like signature-based detection[1, 2, 3] is inadequate in detecting and preventing malware from infecting devices, systems, and networks. This poses a significant threat to critical systems that contain private information. When malware does infect a system, whether it be by exploitation of an open port, un-patched software bug, or user error in terms of a phishing attack, incorrect configuration, etc, the cost can be significant, both monetarily and reputably, that is assuming a security expert can identify the problem and halt the spread of malware, close a backdoor, or reverse an encryption attack from ransomware. In this context, AI-based malware detection approaches pose a powerful and effective approach to defense. They, however, rely on annotated and curated data repositories for their training, evaluation, and updates.

Currently, some datasets enable malware detection through the use of machine learning (ML) such as BODMAS[4], EMBER[5], SOREL-20M[6], and the Microsoft Malware Classification Challenge (BIG 2015)[7]. They allow the user to train a model to identify patterns and behaviors in the features extracted from the malicious and benign samples. These datasets usually contain annotations on malicious samples versus benign samples and information on the malware families and timestamps.

However, none of the datasets currently available to train ML approaches include information on the exploitable vulnerabilities that exist in the malware. The availability of annotated malware samples with known proof of concept (POC) vulnerabilities will not only create ML-based malware detection models

CAMLIS'24: Conference on Applied Machine Learning for Information Security, October 24–25, 2024, Arlington, VA

*Corresponding author.

nross12@qub.ac.uk (N. Ross); o.olukoya@qub.ac.uk (O. Olukoya); j.martinez-del-rincon@qub.ac.uk (J. M. d. Rincón); d.carlin@qub.ac.uk (D. Carlin)

https://pure.qub.ac.uk/en/persons/nathan-ross (N. Ross); https://pure.qub.ac.uk/en/persons/oluwafemi-olukoya (O. Olukoya); https://pure.qub.ac.uk/en/persons/jesus-martinez-del-rincon (J. M. d. Rincón); https://pure.qub.ac.uk/en/persons/domhnall-carlin (D. Carlin)

0009-0001-7324-4837 (N. Ross); 0000-0003-2771-2553 (O. Olukoya); 0000-0002-9574-4138 (J. M. d. Rincón); 0000-0002-8424-2757 (D. Carlin)

but will support the identification of vulnerabilities within them. As an initial step, this approach will not only detect but also stop the spread of malware once it is in the system.

This paper addresses an existing shortfall in malware vulnerability datasets by creating the first comprehensive dataset for static analysis of Windows portable executable (PE) malware vulnerabilities for detection and analysis. Our dataset contains 45,255 samples comprised of malware and benign samples that are labeled vulnerable and non-vulnerable to allow for the development of ML models for detecting malware, vulnerabilities, or both. The creation of this dataset aims to underpin innovative approaches to enhance the security of cyber systems.

The main contributions of this paper are:

- A comprehensive and curated malware vulnerability dataset, which is publicly available[1].
- Extensive annotations on the data including timestamps, families, threat details, vulnerability details, CWEs, packer types and complexities, vulnerability payloads, and Mitre ATT&CK framework mapping.
- A detailed feature set ready to be used that utilizes the well-spread Ember feature extraction[2], which provides greater flexibility to researchers and developers.
- Data visualization of our dataset was performed using dimensionality reduction techniques to depict the data distribution in our dataset.
- Timestamp collection, obfuscation and vulnerability analysis, and CWE and ATT&CK mappings between benign and malware samples, were performed for accurate annotation and richer analysis of the dataset and potential defenses.
- A comprehensive ML benchmark using our dataset for binary (malware vs. benign, vulnerable vs. non-vulnerable) and multi-class classification tasks.

2. Background

In this section, we describe approaches for identifying vulnerabilities within malware in Section 2.1, the application of machine learning techniques for malware classification and vulnerability detection in Section 2.2, and finally, a comparative analysis of relevant datasets used in malware classification and detection, which emphasizes the contribution of our dataset in Section 2.3.

2.1. Malware Vulnerabilities

Research into vulnerabilities within malware is a promising yet under-explored domain. The idea that malware can harbour exploitable flaws makes for an imperative defensive mechanism [8]. In 2010, Caballero et al.[9] introduced a novel approach, termed *stitched dynamic symbolic execution*, aimed at uncovering vulnerabilities within the malware. They aimed to facilitate the discovery of exploitable bugs as a significant step in automated malware analysis. Caballero et al. identified six distinct bugs in four common malware families, highlighting their persistence across multiple evolutions. These weaknesses were exploitable to disrupt or at least hinder their operations defensively. These findings were corroborated in [10], where the authors highlight that malware is prone to exploitable bugs, just like benign software. The authors suggest that most malware does not go through quality control processes, meaning not only are they vulnerable to common issues during the software development lifecycle, but exposed to them in a greater manner. Defensive exploitation of such vulnerabilities can allow defenders to delay, mitigate or frustrate malware-based attacks.

[1] https://github.com/nross12/PEVuln/
[2] https://github.com/elastic/ember

Given that these vulnerabilities exist, they can be identified similarly to how software is labelled in lists such as MITRE Common Weakness Enumeration (CWE)[3]. The studies in [10, 11] also demonstrate that no proper quality assurance is carried out when developing malware since it contains multiple bugs persistent in families across a timeline. This exemplifies how inherited flaws within malware can be exploited defensively. The research in [11] also highlights a bug that crashes Command & Control (C2) servers in the Mirai code, and how the vulnerability persists in many variants. Similar case studies have shown that a DLL-hijack vulnerability was found and exploited in the *LockBit* ransomware [12], which worked against nearly every other ransomware family. This vulnerability has been likened to a Pandora's box of vulnerabilities [13, 14, 15]. Apart from the lack of quality control, code reuse is likely another reason for the persistence of malware vulnerabilities. A study on malware evolution and code reuse over four decades[16] found many instances of code reuse. Findings from research conducted by *Intezer* and *McAfee*, which involved analyzing malware samples and cyber campaigns, revealed substantial evidence of code reuse spanning 10 years (2007-2017) [17]. This investigation helped uncover previously unknown connections among North Korea's malware families, indicating that the reuse of malware code is widespread in cybercrime. The lack of quality control, the rise of malware variants, code reuse, and malware-as-a-service ensure that malware vulnerabilities remain.

One of the well-known cases of using exploitable vulnerabilities in malware for offensive security in detection technology is WannaCry, the biggest ransomware attack in history, which spread within days to more than 250,000 systems in 150 countries and was stopped by registering a web domain found in the malware's code [18]. Once the ransomware checked the URL and found it was active, it was shut down – buying precious time and giving organizations room to update their systems. Such vulnerabilities can often persist in malware and its variants for a long time across different target platforms [11, 19]. Similar studies have identified and exploited flaws in ransomware encryption techniques, assisting victims of ransomware and defending against the threats [20, 21, 22]. Other studies have investigated the identification and exploitation of malware vulnerabilities for covert monitoring of C&C servers through protocol infiltration for botnet disruption and breakdown [23, 19, 24, 25]. To develop an automated system that rapidly uncovers exploitable flaws and defects in malicious software as a kill-switch approach, PE malware machine learning models for vulnerability identification and detection could be designed and utilized, under the assumption that benchmark annotated datasets are available for training.

2.2. Machine Learning

ML has become a cornerstone for malware detection and classification with various techniques being applied by researchers to identify patterns and behaviors that can discern malware from typical benign software [26, 27, 28]. ML models can be trained on features derived from three primary analytical methods: *static*, *dynamic* and *hybrid* analysis [29]. *Static* analysis involves examining an executable without explicitly executing it, making it one of the safest and most efficient methods to obtain information [30]. *Dynamic* Analysis, on the other hand, allows you to gather features based on the output observed by the behavior of the executable during runtime on the system [31]. This method is more intensive on the system and can take much longer to generate data, especially when working with a large dataset. *Hybrid* analysis is a time-consuming and resource-intensive method that combines the strength of static and dynamic techniques to provide a comprehensive understanding of malware samples[32]. Additionally, there are *memory* analysis techniques that create a memory image of the malware during dynamic execution for analysis [19, 33, 34]. This paper proposes specifically the extraction of features by static analysis of PE files.

2.3. Datasets

Machine learning models for malware detection rely heavily on datasets that are curated, annotated and comprehensive. Many benchmark datasets are instrumental for malware detection, which can be seen

[3] https://cwe.mitre.org/

in Table 1, but they lack vulnerabilities and metadata within these datasets on vulnerable executables. Additionally, a comparison of the features, vulnerable samples and metadata proposed in our dataset and other open datasets for PE malware is presented in Table 1.

Table 1
Analysis of the features in each dataset compared to the dataset we present in this paper.

Features	BODMAS [4]	EMBER [5]	SOREL-20M [6]	Microsoft [7]	Our Dataset
# Samples	134,435	2,100,000	19,724,997	10,868	45,255
Threat Details	●	○	●	●	●
Family	●	●	○	●	●
Vulnerable Samples	○	○	○	○	●
Vulnerability Details	○	○	○	○	●
Vulnerability Payloads	○	○	○	○	●
CWE	○	○	○	○	●
Packer (Type/Complexity)	○	○	○	○	●
ATT&CK Techniques	○	○	○	○	●

Note: ○: Absent ●: Present

3. Dataset Description

We created this dataset by systematically collecting data from several malware repositories, data sources, and techniques which are categorized into four main classes as outlined below:

- Vulnerable Malware (VM): The samples and metadata for vulnerable malware were collected from four main sources - Malvuln[4], VirusTotal[5], VulDB[6] and VirusShare[7]. Malvuln is a resource dedicated to malware security vulnerability research and provides information on identifying and exploiting malware, including details such as the threat, vulnerability, description, family, hash, exploit POC, etc. The metadata of each vulnerable malware sample is collected, and the binary executables are downloaded from VirusTotal and VirusShare using the MD5 of the sample and VulDB using the MVID allocated by Malvuln.
- Non-vulnerable Malware(NVM): An abundance of malware was collected, including samples caught by honeypots and from large data dumps on VirusShare. Similarly to the vulnerable malware data, the non-vulnerable malware data contains additional information extracted about each sample from VirusTotal and VirusShare.
- Vulnerable Benign Software(VB): We leveraged *Exploit-DB*[8], which is an exploit database similar to Malvuln, except that it holds labelled conventional software vulnerabilities with options for downloading the vulnerable application, metadata, and exploitation payload.

[4]https://malvuln.com/
[5]https://www.virustotal.com/
[6]https://vuldb.com/
[7]https://virusshare.com/
[8]https://www.exploit-db.com/

- Non-vulnerable Benign Software (NVB): To create a dataset of benign applications, we extracted executables from a fresh installation of Windows 10 (located in *C:\Windows\System32*). This is a popular approach in the malware community for creating a benign dataset [30]. Further vulnerability scans were carried out to ensure that the executables were not vulnerable.

The data sources and types of information collected are summarized in Table 2. Creating an augmentation of an existing dataset or using a combination of multiple sources to create a dataset is consistent with the literature. For instance, an enhanced version of the EMBER dataset, *EMBERSim*, was developed in [35] to include similarity information, addressing the problem of binary code similarity search in Windows PE files. The expanded tags were created using a combination of automatic tagging tools such as AVClass[36, 37] to include class, family, behavior, and file properties. In this research, we utilized AVClass for tagging our data which can provide additional context for addressing the issue of vulnerabilities in PE malware by highlighting the prevalence of specific vulnerabilities across different threat types or families. Table 1 expands on these classes with details on the number of samples in the classes and the number of unique families, vulnerabilities, and CWEs. The vulnerable malware subset is naturally the minority class given how little analysis exists of vulnerabilities in malware samples. This results in a highly imbalanced dataset, and may preclude specific training strategies as we will describe in Section 4.

This dataset will be regularly updated as new vulnerable PE malware and benign samples are collected from the repository.

Table 2
Data Sources and Types of Information Collected.

Source	PE File	MD5/ SHA256	Threat Type	Family	Vulnerability Details	CWE	Mitre ATT&CK
Malvuln	○	●	●	●	●	○	○
ExploitDB	●	●	○	○	●	●	○
VirusTotal	○	●	●	●	○	○	○
VirusShare	●	●	○	○	○	○	○
VulDB	○	●	●	●	●	●	●
Honeypots	●	○	○	○	○	○	○

Table 3
Table showing the Dataset Distribution

Class	# Samples	# Families	# Vulnerabilities	# CWE
VM	864	128	112	35
NVM	35,241	520	N/A	N/A
VB	1,425	N/A	627*	36
NVB	7,905	N/A	N/A	N/A

*Due to the specificity of some vulnerabilities in the Vulnerable Benign class that has been extracted from ExploitDB, the number of unique vulnerability types is fairly high.

3.1. Framework Overview

We have developed a systematic approach, depicted in Figure 1, for the creation of this malware vulnerability dataset, from meticulously seeking out raw data that has gone through various stages of processing and cleaning, followed by ML model training catering to both binary and multi-class classification tasks to obtain preliminary results and data visualization. Firstly, both vulnerable and non-vulnerable malware and benign software executables, with their corresponding metadata, were sourced and subsequently pre-processed, ensuring that clean and well-structured outputs capture the attributes and features necessary to perform a deeper analysis of the performance of the dataset. In the executable

pre-processing phase, feature extraction is performed using Ember[5] to derive meaningful features from the executables for each class. Obfuscation scanning and static application security testing (SAST) were performed to obtain additional information about the executables. Simultaneously, metadata pre-processing focuses on the supplementary data associated with the executables. This involves mapping hashes such as md5 and sha256 where appropriate so that the samples can be identified and extracting timestamps, threat types, families, and various vulnerability details (such as the vulnerability type and the payload to exploit it). Additionally, the Mitre ATT&CK framework and common weakness enumeration (CWE) were mapped, and obfuscation details (such as the types of packer used) were analyzed and categorized based on complexity. To generate an ML benchmark for the community, as well as to prove useful models can be generated from our dataset, a comprehensive set of common machine learning algorithms were trained, including LightGBM (LGBM), Random Forest (RF), k-Nearest Neighbors (KNN), Support Vector Machines (SVM), and Artificial Neural Networks (ANN). These models are trained to perform both binary and multi-class classification tasks to distinguish between benign and malware data and further categorize if it is vulnerable or not. Full details on the ML baseline models are given in Section 4. Finally, we also visualize our dataset and its analysis in Section 3.7. This presents detailed visualizations of the full dataset by applying dimensionality reduction techniques, namely Principal Component Analysis (PCA) and t-SNE. These techniques allow us to produce 2D representations of our large dimensional data. Data samples are depicted using different labels and colors according to the previously mentioned metadata including vulnerabilities, obfuscation details, CWEs, etc. Furthermore, we have visualized the Mitre ATT&CK mappings to better understand the attack surfaces in which the vulnerable malware targets develop relationships between these attack surfaces and the vulnerabilities present.

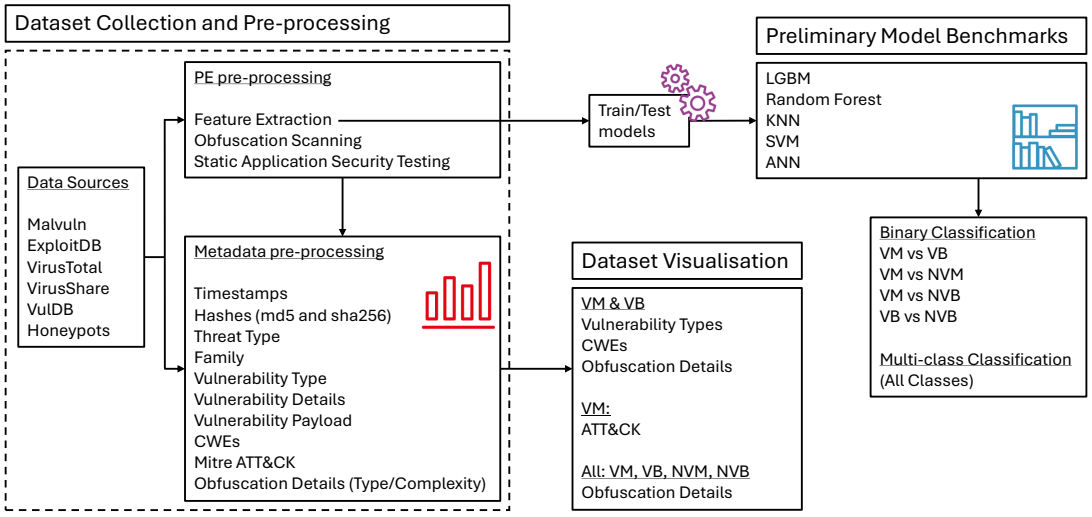

Figure 1: Framework showing the steps to produce this dataset and how preliminary benchmark results were generated with common classifier implementations.

3.2. SAST Tools Scanning

When working with ML, the data needs to be curated and precisely annotated to avoid outliers, training discrepancies and even potential poisoning and/or backdoor attacks. In our case, this is especially relevant regarding the vulnerabilities that could theoretically exist in the non-vulnerable data as it had not gone through rigorous testing in the compilation of this dataset. So for that, Source Code

Analysis Tools (or Static Application Security Testing, SAST) were used on the non-vulnerable class executables to find potential security flaws through the means of signature-based pattern matching, semantic analysis, and taint analysis. This allows for the removal of data that can be ruled out as vulnerable from the non-vulnerable classes which, in theory, should make the vulnerable classes more prominent when training models. To perform this SAST on the data, two tools were used; cve-bin-tool[9] and Vulnscan[10].

3.3. Packing in Vulnerable & Non-vulnerable Malware

As we are working with executables, particularly user software and malware, they can be affected by evasion techniques, more specifically in our case *obfuscation*. To check for this, we used two tools: *Bintropy*[38], a Python tool that detects obfuscation based on entropy, and *Detect It Easy (DIE)*[11] which works similarly to *Bintropy*[12] but also provides information on the packer, linker, and compiler. It is possible to categorize the different types of packers based on their complexities which can be seen in [39]. The types of packers range from Type-I to Type-VI, with higher values indicative of a greater complexity. This method of categorization provides additional information which allows us to conduct a more nuanced analysis of the different packers used in malware compared to benign software. This is critical as it allows an investigation into whether malicious actors lean towards using more complex packers with specialized techniques over others for evasion. These patterns could also inform us of a pattern or correlation between the use of specific packers in malware with vulnerabilities compared to those with no vulnerabilities.

Table 4
Different Packers used on Malware samples

Packer Type	Packer	# V-Malware	# NV-Malware
Type-I	UPX	107	2570
Type-III	ASPack, ASProtect, FSG, NSPack, PE Compact, Upack	43	2504
Unknown	ezip, MEW, MoleBox, MPRESS, NeoLite, Petite, PKLITE, RLPack, DXPack, kkrunchy, PyInstaller, Packman, NakedPacker, SpoonStudio, BeRo, KByS, ASPPack, nPack, JDPack, .NETZ	13	2363
# Unique Packers		14	25
# Packed		398	20998

As shown in Table 4 and Table 5, the most often used packers in the vulnerable and control classes of the benign and malicious samples are rather simple, spanning from Type-I to Type-III based on the packer taxonomy proposed by Ugarte-Pedrero et al.[39]. This packer distribution result is consistent with multiple longitudinal studies [39, 19, 40, 30] investigating the complexity of custom and off-the-shelf run-time packers in the wild. The implication of this study is two-fold. Firstly, packer complexity in control (non-vulnerable) and vulnerable malware follows the same pattern, making it possible to use analysis and detection techniques for *packed* malware to *packed* vulnerable malware. Secondly, there is an overlap between packers used in vulnerable and controlled malware samples, so we can

[9] https://github.com/intel/cve-bin-tool
[10] https://github.com/zhutoulala/vulnscan
[11] https://github.com/horsicq/Detect-It-Easy
[12] https://github.com/packing-box/bintropy

Table 5
Different Packers used on Benign samples

Packer Type	Packer	# V-Benign	# NV-Benign
Type-I	UPX	127	1
Type-III	ASPack, NOS Packer, PECompact	11	0
Unknown	Petite, PKLITE	4	0
# Unique Packers		6	1
# Packed		2193	3400

develop classifiers that do not associate specific packers with vulnerability. This finding is significant, as research on the impact of machine-learning-based malware detection on packed samples utilizing static analysis features has observed that classifiers frequently link particular packers to malicious activity because there is insufficient overlap between packers utilized in malicious and benign samples [30]. Additionally, observations of the count of unique packers in the control benign are restricted to one, which can be attributed to how we collected the data. That being, Windows 10 likely does not need to implement various packers as their executables and DLLs are distributed and managed with the Operating System (OS), and their consistent employment of UPX for all packed executables and DLLs is likely driven by several factors[13]; mainly concerning maintaining consistency and widespread applicability across various executables and DLL files. We can therefore conclude that packers are not a feature for determining the nature of an executable as controlled or vulnerable malware.

3.4. CWE Mapping

For richer vulnerability analysis, we have mapped the CWEs for both the vulnerable malware and vulnerable benign classes in this dataset and conducted strategic similarity analysis between them so that we can understand the crossover between vulnerabilities present in benign software or malware. By doing this, we can further analyze the correlation between the vulnerabilities with other attributes such as system architecture, programming languages used, attack vectors, or exploitation complexities. Mapping CWEs in vulnerable malware is also crucial for linking vulnerabilities across certain malware families to demonstrate both the persistence of vulnerabilities in newer malware of the same family and the possible introduction of newer vulnerabilities. Another advantage of CWE mapping for malware vulnerabilities is that it enables the creation of automated exploits based on the identified weakness category[41]. This is similar to the automated exploit generation for vulnerabilities seen in commercial and open-source programs, such as Stack-based buffer overflow(*CWE-121*)[42, 43], PHP object injection vulnerabilities(*CWE-502*, *CWE-915*)[44] and XML injection vulnerabilities(*CWE-91*)[45].

In Table 6, an analysis of comprehensive weakness categories is presented for vulnerable malicious and benign samples. The analysis reveals that the top 10 most frequent vulnerability categories in the vulnerable malware dataset have a 20% similarity with the top 10 most frequent vulnerabilities in the vulnerable benign dataset, with CWE-200 and CWE-120 being the common vulnerability categories. This suggests that both classes contain vulnerabilities that can lead to information exposure and buffer overflow. The findings provide insight into the categories of vulnerabilities present in vulnerable malware and their prevalence. Additionally, the analysis highlights the overlap and differences between the vulnerabilities in legitimate software applications and malicious binaries, indicating that vulnerable malicious binaries often exhibit different categories of weaknesses compared to their benign counterparts.

Approximately 71.69% of vulnerabilities in the malware samples are attributed to five main classes of weaknesses. These classes include *Permission Issues*, *Hidden Functionality*, *Buffer Overflows*, *Improper*

[13] https://upx.github.io/

Authentication, and *Use of Hard-coded Credentials*. They have consistently been the primary sources of vulnerabilities, making them favoured targets for defenders seeking to exploit these security issues. The *Permission Issues* category refers to weaknesses related to improper assignment or handling of permissions. A comprehensive study on covert monitoring of C&C servers [19] found that *over-permissioned* protocols are prevalent in the malware operational landscape, with nearly 1 in 3 malware bots exhibiting this vulnerability, confirming the significance of this weakness category. Another study on IoT botnets [23] highlighted weak and default passwords in C2 servers, aligning with the identification of *Improper Authentication* and *Use of Hard-coded Credentials* as primary sources of malware vulnerabilities. It is also expected that CWE-912, *Hidden Functionality*, is prevalent in our dataset, as it can take the form of embedded malicious code and is useful for attacks that modify the control flow of the application, aligning with malicious behavior. Another noteworthy weakness category in the Top 10 for malware is CWE-426, a weakness category associated with DLL Hijacking, which is prevalent and exploitable in most ransomware families for preventing file encryption [13, 14, 15].

In benign executables, 76.64% of vulnerabilities are attributed to two main classes of weaknesses. The primary sources of vulnerabilities are *Improper Restriction of Operations within the Bounds of a Memory Buffer* and *Improper Input Validation*. These weaknesses are commonly targeted by threat actors when trying to exploit security issues. Our ranking, based on our dataset, is further supported by the 2023 CWE Top 10 KEV weaknesses[14], which is a catalogue of Known Exploited Vulnerabilities maintained by the Cybersecurity and Infrastructure Security Agency (CISA)[15] for vulnerability management prioritization. Notably, four of the Top 10 CWEs identified in the vulnerable benign dataset, CWE-787 (#3), CWE-20 (#4) and CWE-22 (#9), are among the top 10 class of weaknesses exploited by threat actors as observed in the wild. This suggests that the vulnerabilities and prevalence identified in the dataset accurately reflect real-world observations.

We have further conducted a detailed analysis of the weakness categories present in the vulnerable samples by aligning the weaknesses with the OWASP Top 10[16] and the 2023 CWE Top 25[17], as shown in Table 7. The identified malware CWEs correspond to 8 of the OWASP Top 10 application security risks, whereas benign CWEs correspond to 6 categories. Similarly, the malware CWEs align with 9 of the top 25 most dangerous software weaknesses, while benign CWEs align with 8 categories. The CWE classifies vulnerabilities in benign and malicious binaries into the same four categories in the OWASP Top 10. This indicates that vulnerability detection and identification tools developed for these categories in benign software may apply to malicious binaries due to the overlap. For example, the technique of symbolic execution, used in TaintScope[46] to find bugs in benign software, was also applied to identify bugs in prevalent families of bots and other malware in a study by Caballero et al[9]. Moreover, the CWE from both classes sometimes maps to unique categories of critical risk in the OWASP Top 10 without overlapping. This supports the initial assertion about the differences in vulnerabilities in both samples. For instance, only the malware samples have vulnerabilities associated with A02:2021 - Cryptographic Failures. One characteristic of this category is a failure in the encryption mechanism, which is significant as one common vulnerability exploited in ransomware is encryption failures[21, 22]. When the first ransomware bug bounty operation was launched, Locker Bugs, which covers encryption errors, was prioritized[12]. This analysis demonstrates that although there is awareness of software vulnerabilities, it is essential to recognize that malware possesses inherent vulnerabilities that are critical and impactful. These vulnerabilities can be exploited for offensive security in defense technology.

[14] https://cwe.mitre.org/top25/archive/2023/2023_kev_list.html
[15] https://www.cisa.gov/known-exploited-vulnerabilities-catalog
[16] https://owasp.org/www-project-top-ten/
[17] https://cwe.mitre.org/top25/index.html
[18] https://nvd.nist.gov/vuln/categories

Table 6
Number of observations per CWE in our vulnerable malware and benign dataset using the NVD CWE Slice [18]

CWE in Malware	CWE Description	Count	CWE in Benign	CWE Description	Count
CWE-275	Permission Issues	133	CWE-119	Improper Restriction of Operations within the Bounds of a Memory Buffer	452
CWE-912	Hidden Functionality	106	CWE-20	Improper Input Validation	50
CWE-120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	40	CWE-94	Improper Control of Generation of Code ('Code Injection'	21
CWE-287	Improper Authentication	35	CWE-399	Resource Management Errors	20
CWE-121	Stack-based Buffer Overflow	31	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	20
CWE-798	Use of Hard-coded Credentials	27	CWE-189	Numeric Errors	17
CWE-404	Improper Resource Shutdown or Release	18	CWE-787	Out-of-bounds Write	12
CWE-259	Use of Hard-coded Password	18	CWE-120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	10
CWE-426	Untrusted Search Path	17	CWE-200	Exposure of Sensitive Information to an Unauthorized Actor	6
CWE-200	Exposure of Sensitive Information to an Unauthorized Actor	11	CWE-264	Permissions, Privileges, and Access Controls	5
CWE-122	Heap-based Buffer Overflow	10	CWE-416	Use After Free	5
CWE-319	Cleartext Transmission of Sensitive Information	10	CWE-358	Improperly Implemented Security Check for Standard	5
CWE-428	Unquoted Search Path or Element	9	CWE-352	Cross-Site Request Forgery (CSRF)	4
CWE-284	Improper Access Control	8	CWE-611	Improper Restriction of XML External Entity Reference	3
CWE-306	Missing Authentication for Critical Function	8	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	3
CWE-119	Improper Restriction of Operations within the Bounds of a Memory Buffer	6	CWE-428	Unquoted Search Path or Element	2
CWE-427	Uncontrolled Search Path Element	4	CWE-36	Absolute Path Traversal	2
CWE-77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	3	CWE-824	Access of Uninitialized Pointer	2
CWE-21	DEPRECATED: Pathname Traversal and Equivalence Errors	3	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	2
CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	3	CWE-476	NULL Pointer Dereference	2
CWE-476	NULL Pointer Dereference	3	CWE-134	Use of Externally-Controlled Format String	2
CWE-918	Server-Side Request Forgery (SSRF)	2	CWE-287	Improper Authentication	1
CWE-312	Cleartext Storage of Sensitive Information	2	CWE-918	Server-Side Request Forgery (SSRF)	1
CWE-300	Channel Accessible by Non-Endpoint	2	CWE-835	Loop with Unreachable Exit Condition ('Infinite Loop')	1
CWE-352	Cross-Site Request Forgery (CSRF)	2	CWE-763	Release of Invalid Pointer or Reference	1
CWE-23	Relative Path Traversal	2	CWE-732	Incorrect Permission Assignment for Critical Resource	1
CWE-256	Plaintext Storage of a Password	1	CWE-770	Allocation of Resources Without Limits or Throttling	1
CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	1	CWE-190	Integer Overflow or Wraparound	1
CWE-434	Unrestricted Upload of File with Dangerous Type	1	CWE-522	Insufficiently Protected Credentials	1
CWE-521	Weak Password Requirements	1	CWE-121	Stack-based Buffer Overflow	1
CWE-288	Authentication Bypass Using an Alternate Path or Channel	1	CWE-88	Improper Neutralization of Argument Delimiters in a Command ('Argument Injection')	1
CWE-313	Cleartext Storage in a File or on Disk	1	CWE-427	Uncontrolled Search Path Element	1
CWE-759	Use of a One-Way Hash without a Salt	1	CWE-255	Credentials Management Errors	1
CWE-611	Improper Restriction of XML External Entity Reference	1	CWE-269	Improper Privilege Management	1
CWE-314	Cleartext Storage in the Registry	1	NVD-CWE-Other*	Other	291
*NVD does not use the whole CWE for mapping; instead, it uses a subset of CWE that does not include the weakness type. ** Insufficient information to classify the issue caused by unknown or unspecified details.			NVD-CWE-noinfo**	Insufficient Information	11

3.5. MITRE ATT&CK Mapping

The Mitre ATT&CK framework is essential in understanding and leveraging tactics and techniques for both vulnerability exploitation and defense mechanisms. Tactics[19] within the ATT&CK framework represent the strategic objectives or the "why" behind an adversary's actions, whereas techniques[20] detail the "how" by describing the specific methods used to accomplish these tactical goals. With this

[19] https://attack.mitre.org/tactics/enterprise/
[20] https://attack.mitre.org/techniques/enterprise/

Table 7
CWEs from Vulnerable Malware (M) and Benign samples (B) mapped to the OWASP Top 10 and CWE Top 25 Lists.

OWASP Top 10	CWE (Benign)	CWE (Malware)	2023 CWE Top 25
A01:2021 Broken Access Control	CWE-22 CWE-200 CWE-264 CWE-352	CWE-23 CWE-200 CWE-275 CWE-284 CWE-352	(B) CWE-22 (#8) (B) CWE-352 (#9) (M) CWE-352 (#9)
A02:2021 Cryptographic Failures		CWE-319 CWE-759	
A03:2021 Injection	CWE-20 CWE-79 CWE-88 CWE-89 CWE-94	CWE-77 CWE-79 CWE-89	(B) CWE-20 (#6) (M) CWE-77 (#16) (B) CWE-79 (#2) (M) CWE-79 (#2) (B) CWE-89 (#3) (M) CWE-89 (#3) (B) CWE-94 (#23)
A04:2021 Insecure Design	CWE-269 CWE-522	CWE-256 CWE-312 CWE-313 CWE-434	(B) CWE-269 (#22) (M) CWE-434 (#10)
A05:2021 Security Misconfiguration	CWE-611	CWE-611	
A07:2021 Identification and Authentication Failures	CWE-255 CWE-287	CWE-259 CWE-287 CWE-288 CWE-300 CWE-306 CWE-521 CWE-798	(B) CWE-287 (#13) (M) CWE-287 (#13) (M) CWE-306 (#20) (M) CWE-798 (#18)
A08:2021 Software and Data Integrity Failures		CWE-426	
A10:2021 Server Side Request Forgery	CWE-918	CWE-918	(B) CWE-918 (#19) (M) CWE-918 (#19)

knowledge, potential attack paths in malware can be established and exploited, or conversely can be used for counteracting adversarial actions in benign software. In our dataset, these tactics and techniques are mapped according to the vulnerabilities in the vulnerable malware class, which can be seen in Table 8.

Table 8 shows the top ATT&CK Tactics and Techniques adopted in the vulnerable malware, which are overwhelmingly File and Directory Permissions Modification, Obtain Capabilities (malware), Brute Force (Password Guessing), Hijack Execution Flow and Valid Accounts (Default Accounts) and makes up over 82% of the techniques. These techniques show that the adversaries want to infiltrate the victim's network, evade detection throughout the compromise process, establish resources they can use to support operations, steal account credentials, obtain higher-level permissions and maintain their foothold. It is important to note that the tactics and techniques in Table 8 are only representative of our dataset, which is influenced by the number of malware families and samples. Under different conditions, the order will be different. In any case, the present tactics and techniques of vulnerable malware are the ones you would expect from malware generally [47, 48], which shows that the vulnerability in malware

Table 8
Mitre Tactics and Techniques, mapped to their description with the number of times they appear in the dataset mapped to the Vulnerable Malware class.

# Total Techniques	Technique ID	Tactics
133	T1222 File and Directory Permissions Modification	TA0005 Defense Evasion
106	T1588.001 Obtain Capabilities (malware)	TA0042 Resource Development
27	T1110.001 Brute Force (Password Guessing)	TA0006 Credential Access
21	T1574 Hijack Execution Flow	TA0003 Persistence
		TA0004 Privilege Escalation
		TA0005 Defense Evasion
18	T1499 Endpoint Denial of Service	TA0040 Impact
18	T1078.001 Valid Accounts (Default Accounts)	TA0001 Initial Access
		TA0003 Persistence
		TA0004 Privilege Escalation
		TA0005 Defense Evasion
11	T1592 Gather Victim Host Information	TA0043 Reconnaissance
10	T1040 Network Sniffing	TA0006 Credential Access
		TA0007 Discovery
9	T1574.009 Hijack Execution Flow (Path Interception by Unquoted Path)	TA0003 Persistence
		TA0004 Privilege Escalation
		TA0005 Defense Evasion
8	T1068 Exploitation for Privilege Escalation	TA0004 Privilege Escalation
5	T1006 Direct Volume Access	TA0005 Defense Evasion
4	T1555 Credentials from Password Stores	TA0006 Credential Access
3	T1202 Indirect Command Execution	TA0005 Defense Evasion
3	T1059.007 Command and Scripting Interpreter (JavaScript)	TA0002 Execution
2	T1557 Adversary-in-the-Middle	TA0006 Credential Access
		TA0009 Collection
2	T1552 Unsecured Credentials	TA0006 Credential Access
1	T1608.002 Stage Capabilities (Upload Tool)	TA0042 Resource Development
1	T1110.002 Brute Force (Password Cracking)	TA0006 Credential Access
1	T1505 Server Software Component	TA0003 Persistence

is not some sophistication of tactics and techniques. Similarly to every piece of software code, malware is also prone to vulnerabilities and weaknesses that can be exploited. To find the TTPs that were present in over 600,000 malware samples that were gathered between January 2023 and December 2023, Picus Labs [48] conducted an analysis. The study identified the top 10 MITRE ATT&CK techniques, which differ from the top 10 ATT&CK techniques used in this study's vulnerable malware samples. A related study that looked into ATT&CK Trends and Techniques in 951 Windows malware families between 2017 and 2018 revealed that the examined dataset had a varied number of observations for each ATT&CK [47]. For example, only three of the dataset's top ten ATT&CK approaches were adopted by threat groups and malware, according to the report by Picus Labs. Top ATT&CK techniques can vary depending on the specific malware family being considered, as different families have different why and how due to their tactical goals and actions. For instance, the techniques used by ransomware will differ from those used by spyware. For example, the Centre for Threat Informed Defense, MITRE Engenuity [49], analyzed 22 ransomware groups over 3 years and compiled a list of the Top 10 ATT&CK techniques. As anticipated, the Top 10 ATT&CK Techniques for ransomware only had three techniques (*T1486 - Data Encrypted for Impact, T1027 - Obfuscated Files or Information, T1055 - Process Injection*) in common with the top ATT&CK techniques identified by Picus Labs [48] and the trends in Windows Malware [47].

Vulnerable malware is still malware and adheres to the same characteristics and propagation strategies. Therefore, its vulnerabilities can persist in malware families and variants for a long time because the authors either do not recognize the vulnerabilities, have poor operational security, or just lack a quality control process and are more likely to have a bug that can persist in malware families and variants for a long time, which persists as a notable observation that we have highlighted by Singh in [10]. Vulnerable or not, malware tactics and techniques are still adversarial. To prioritize ATT&CK techniques for defending against malware attacks, creating a top ATT&CK techniques list in Table 8 can be a

helpful starting point. This approach is similar to MITRE's methodology[21], which takes into account the prevalence of techniques, common attack choke points, and actionability to help defenders focus on the most relevant techniques for their organization. Including the number of observations per technique allows us to measure how frequently an attacker uses a specific MITRE ATT&CK technique, which can be useful in identifying important techniques when dealing with vulnerable malware for exploitation. Additionally, defenders have the opportunity to mitigate or defend against each ATT&CK based on publicly available threat intelligence derived from real-world observations. MITRE ATT&CK offers additional information for mitigation, detection, procedure examples, and references for each identified significant technique, which can serve as a knowledge base for offensive security and detection technology.

3.6. EMBER Feature Set

For the data to be used with ease as a benchmark for classification models, Ember, an open-source dataset and feature extraction research project that uses the LIEF[22] project to extract features from PE files for static malware detection, was utilized to systematically generate a comprehensive set of vectorized features. The resulting feature vector per sample contains 2381 elements, using the version 2 extractor, from the PE files in our data. This feature set is widely applied in the ML malware detection literature [35]. Thus, the extraction of static EMBER features from our dataset is consistent with other benchmark datasets for malicious PE detection in the literature[4, 6]. For the features extracted using Ember, data normalization is applied to allow for the values between the different classes to be distributed across a common scale. This allows us to work with data within the desired range without losing much of the original distribution. The *MinMaxScaler* is the method used to normalize input features to the [0, 1] range based on the training data. This normalization technique has been applied for feature embedding when working with other benchmark PE malware datasets such as the EMBER dataset to enhance ML training [50, 51].

3.7. Visualizations of Data Distributions

To visualize the data distribution and have a better understanding of our dataset, Principal Component Analysis (PCA) was applied. PCA is a statistical method that reduces the dimensionality of large datasets, in our case our complexity was derived from the 4 classes and a vector length of 2381 from the Ember features, by transforming them into smaller linear and uncorrelated variables, known as principal components. By reducing the dimensionality to only 2 variables (the two first principal components), the dataset can be visualized as an image. Similarly, t-distributed Stochastic Neighbor Embedding (t-SNE)[52] was also used, by employing an unsupervised, non-linear reduction of dimensionality. However, unlike PCA, t-SNE preserves the local structure of the data which allows for more visible clusters.

From Figure 2, it is clear that there is a discernible separation between the malware and benign software classes. Notably, there is an overlap between the vulnerable and control malware. This is to be expected as the patterns and behavior in malware should be consistent between samples. The same is expected and seen in the vulnerable and control benign samples. There is, however, an existing overlap between the vulnerable malware and vulnerable benign samples. This raises the assumption that PCA is categorizing these two classes based on what we hope to be their vulnerabilities.

The analysis derived from Figure 2 translates well to Figure 3. However, there is even more discernible separation, especially between the different samples within each class, seen from the prominent clusters formed. Similar to before, the malware and benign software classes are separated. However, the grouping for vulnerable malware and vulnerable benign software is more noticeable, which again we can assume to be features that capture their vulnerabilities.

[21] https://top-attack-techniques.mitre-engenuity.org/methodology
[22] https://github.com/lief-project/LIEF

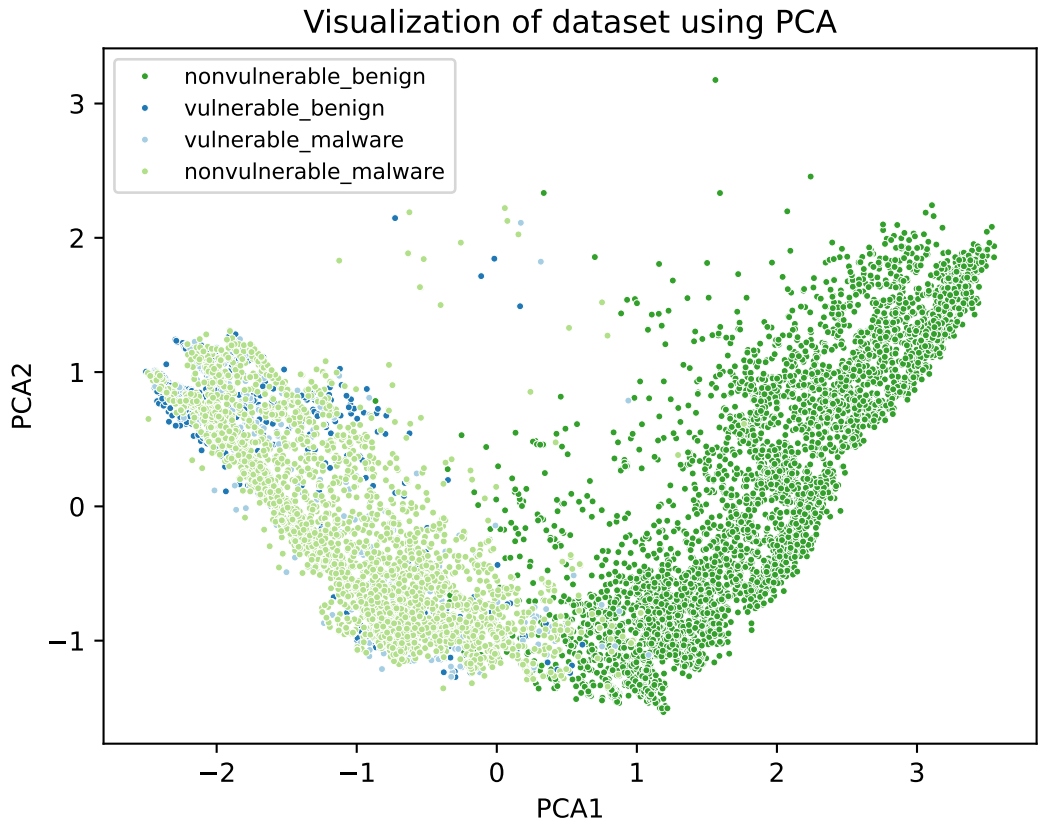

Figure 2: Visualization of the data distribution using PCA as a scatterplot.

From those visualizations, it can be concluded that our dataset represented as Ember feature vectors will allow for the effective separation of malware and benign samples and the training of a malware detector. On the contrary, the identification of vulnerabilities within the program may result in a more challenging problem.

4. Model Benchmarks

4.1. Model Setup

As previously described, a comprehensive set of common ML-based models were used, including LightGBM (LGBM)[23], Random Forest (RF)[24], K-Nearest Neighbors (KNN)[25], Support Vector Machine (SVM)[26], and an Artificial Neural Network (ANN), specifically a Multi-layer Perceptron (MLP)[27]. By using these models, which operate on different principles and assumptions, we can observe distinct perspectives on the dataset and provide a comprehensive set of baseline models that the scientific community can use to benchmark their novel approaches.

In our experimental setup, these classifiers were trained initially with an 80:20 train and test split. Additionally, we implemented stratified K-fold[28] as a way to mitigate any overfitting that may exist in our models and provide a more comprehensive and realistic evaluation. The reasoning for using a

[23] https://lightgbm.readthedocs.io/en/stable/
[24] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
[25] https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
[26] https://scikit-learn.org/stable/modules/svm.html
[27] https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html
[28] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html

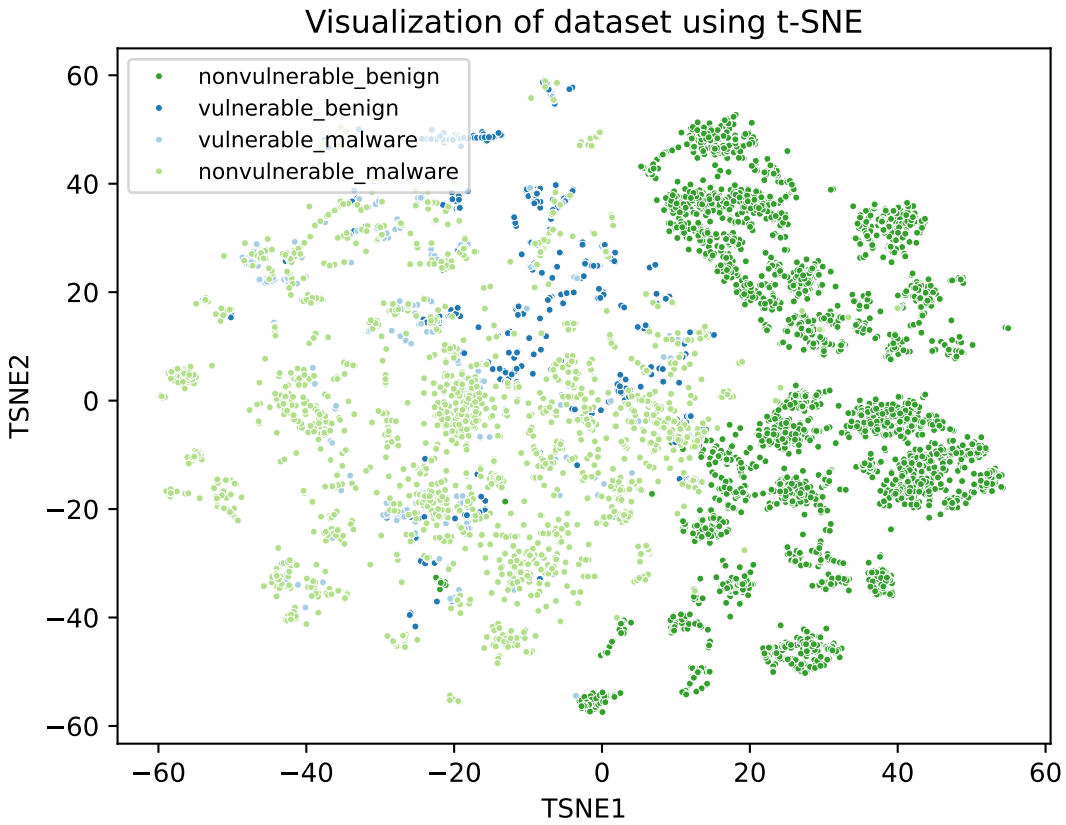

Figure 3: Visualization of the data distribution using t-SNE as a scatterplot.

stratified K-fold ensures that each fold maintains approximately the same percentage of samples for each target class in the entire dataset, especially considering the class imbalance that exists where the vulnerable malware class is underrepresented. We used a k-fold of "K=5" and the cross-validation model provides train/test indices to split data into train/test sets as a default. After the results were calculated, we took a mean of each estimator to obtain the final values. Performance metrics such as accuracy, weighted F1-score, weighted precision, and weighted recall were calculated to evaluate the efficacy of the models on both default parameters and hyperparameter optimization. As for stratified K-fold, the accuracy metric was replaced with balanced_accuracy, defined as the average of recall obtained on each class, specifically designed for dealing with unbalanced classes.
A set of ten different tasks were addressed and validated using our dataset, which includes:

- Vulnerable Malware vs Non-vulnerable Malware in Table 12
- Vulnerable Malware vs Vulnerable Benign in Table 13
- Vulnerable Malware vs Non-vulnerable Benign in Table 14
- Vulnerable Benign vs Non-vulnerable Benign in Table 15
- Vulnerable Benign vs Non-vulnerable Malware in Table 16
- Non-vulnerable Benign vs Non-vulnerable Malware in Table 17
- Vulnerable Malware vs All Benign (VB+NVB) in Table 18
- Vulnerable Benign vs All Malware (VM+NVM) in Table 19

- Malware (VM+NVM) vs Benign (VB+NVB) in Table 10
- Multi-class (All four classes: VM, NVM, VB and NVB) in Table 11

Hyperparameter optimization was applied to provide the best possible performance for each machine learning technique. While some studies use default settings and parameters, such as those in EMBER[5], a better approach involves extensive hyperparameter tuning, as seen in the UCSB Packed Malware dataset[30]. Our model parameter settings align with these approaches. Subsequently, we implemented hyperparameter optimization using gridsearch[29] for all models. These parameters can be seen in Table 9. Additionally, we used the optimized parameters when implementing the estimators for cross validation.

Table 9
Parameters used for both multi-class and binary tasks obtained from Hyperparameter Optimization

Classifier	Multi-class	Binary
LGBM	'bagging_fraction': 0.8, 'feature_fraction': 0.9, 'learning_rate': 0.1, 'max_bin': 20, 'max_depth': 30, 'min_data_in_leaf': 20, 'min_sum_hessian_in_leaf': 0, 'n_estimators': 200, 'num_leaves': 24, 'objective': 'multiclass', 'subsample': 0.01	'bagging_fraction': 0.8, 'feature_fraction': 0.9, 'learning_rate': 0.1, 'max_bin': 20, 'max_depth': 5, 'min_data_in_leaf': 80, 'min_sum_hessian_in_leaf': 0, 'n_estimators': 200, 'num_leaves': 80, 'subsample': 0.01
RF	'bootstrap': False, 'max_depth': 70, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 800	'bootstrap': False, 'max_depth': 30, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 400
KNN	'metric': 'manhattan', 'n_neighbors': 5, 'weights': 'distance'	'metric': 'minkowski', 'n_neighbors': 5, 'weights': 'distance'
SVM	'C': 100, 'gamma': 'scale', 'kernel': 'rbf'	'C': 100, 'gamma': 'scale', 'kernel': 'rbf'
ANN	'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': (100,), 'learning_rate': 'constant', 'solver': 'adam'	'activation': 'relu', 'alpha': 0.001, 'hidden_layer_sizes': (50,), 'learning_rate': 'adaptive', 'solver': 'adam'

4.2. Binary Classification

Table 10
Benchmarks on different models for a binary malware detection task. The classes used were all malware samples vs all benign samples.

Technique	Model	Accuracy	F1-Score	Precision	Recall
Default Parameters	LGBM	0.984	0.984	0.984	0.984
	RF	0.977	0.977	0.977	0.977
	KNN	0.979	0.979	0.979	0.979
	SVM	0.949	0.949	0.951	0.949
	ANN	0.976	0.976	0.976	0.976
Hyperparameter Optimization	LGBM	0.983	0.983	0.983	0.983
	RF	0.980	0.980	0.980	0.980
	KNN	0.981	0.981	0.981	0.981
	SVM	0.980	0.980	0.980	0.980
	ANN	0.974	0.974	0.975	0.974
Technique	Model	Balanced Accuracy	F1-Score	Precision	Recall
Stratified K-Fold	LGBM	0.952	0.950	0.963	0.952
	RF	0.951	0.950	0.963	0.952
	KNN	0.949	0.947	0.961	0.949
	SVM	0.948	0.947	0.960	0.948
	ANN	0.941	0.939	0.953	0.941

Firstly, combining all classes into a binary task (malware vs. benign) allows us to benchmark our dataset against others that exist in [4, 5, 6, 7] for malware detection. This not only aligns with the conventional practices seen in the literature but also simplifies this classification task so that we can analyze a more straightforward model performance.

[29] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

4.3. Multi-class Classification

Table 11
Benchmarks on different models using default parameters, hyperparameter optimization and stratified K-Fold cross-validation with the dataset for a multi-class classification task.

Technique	Model	Accuracy	F1-Score	Precision	Recall
Default Parameters	LGBM	0.957	0.955	0.956	0.957
	RF	0.940	0.934	0.936	0.940
	KNN	0.942	0.942	0.940	0.942
	SVM	0.918	0.898	0.879	0.918
	ANN	0.934	0.925	0.931	0.934
Hyperparameter Optimization	LGBM	0.960	0.958	0.958	0.960
	RF	0.940	0.933	0.938	0.940
	KNN	0.935	0.933	0.931	0.935
	SVM	0.948	0.946	0.945	0.948
	ANN	0.938	0.936	0.934	0.938
Technique	Model	Balanced Accuracy	F1-Score	Precision	Recall
Stratified K-Fold	LGBM	0.855	0.956	0.957	0.958
	RF	0.790	0.935	0.939	0.942
	KNN	0.830	0.941	0.940	0.943
	SVM	0.843	0.948	0.948	0.951
	ANN	0.822	0.939	0.940	0.943

Subsequently, we can compare the results from Table 10 with the results we have obtained in Table 11 to discern between how the ML classifiers distinguish between an added complexity of vulnerabilities.

Across all tasks and techniques, the LGBM model consistently outperformed others, achieving an average F1-Score of between 0.929 and 0.999. Albeit, all models demonstrated a strong performance, especially considering the notable variance in class sizes and similar data structure between the related classes, that being either the vulnerable and non-vulnerable malware or benign classes. This tells us that these models can find patterns in the data that can recognise if a sample is either **1)** malware or benign and **2)** vulnerable or non-vulnerable. One alternative method for establishing model benchmarks could involve utilizing representation learning[53, 50, 54, 55]. This approach aims to differentiate samples that share similarities in the feature space. This could be a viable research direction as the t-SNE plots in Figure 3 show some overlap among the VM, VB, and NVM classes.

5. Discussion and Conclusion

5.1. Finding Vulnerabilities in Malware: Applications and Ethical Considerations

The public disclosure of vulnerabilities in commercial and open-source programs has sparked conversations and led to the creation of responsible disclosure programs. These programs provide guidelines for reporting and cataloguing vulnerabilities. For instance, CISA KEV[56] is a catalogue of Known Exploited Vulnerabilities that updates its entries only when a vulnerability has been assigned a common identifier for publicly known cybersecurity vulnerability, actively exploited, and there is clear remediation guidance. On the other hand, VulnCheck KEV[57] adds vulnerabilities to their catalogue as long as the vulnerability is publicly reported as exploited in the wild. Unlike CISA KEV, VulnCheck KEV does not have additional criteria such as an identifier and clear remediation guidance. It is still unclear whether there might be a similar requirement for vulnerability disclosure in malware. Currently, the Malvuln project is leading that charge by cataloguing vulnerabilities found in malware and providing exploitation details.

The dataset and benchmarks in this research focus on using exploitable vulnerabilities in malware for offensive security in defense technology, rather than public disclosure of malware vulnerabilities. While it is known that attackers exploit vulnerabilities in benign software, we are looking at the opposite scenario, where defenders can identify vulnerabilities in malware to enhance threat intelligence for cyber defense. The dataset aims to advance research on using exploitable malware vulnerabilities as a defense layer against malware.

5.2. Future Work

The contributions to the POC vulnerabilities in malware within this dataset are supplied only by Malvuln. This single-author database, although regularly updated, requires subsequent future work to aid in obtaining samples to investigate the impact of the sample size on the performance metrics which is necessary for the research of vulnerability detection in malware.

One potential enhancement from Section 3.4 is to the CWE mapping of malicious binaries and the incorporation of CWE chains and composite[58]. This involves identifying the relationships between different CWEs in a vulnerable sample, which can be implicit, named or composite as defined by MITRE's knowledge base. By identifying these relationships, we can understand how weaknesses can be combined to create vulnerabilities which can then inform the process of generating automated exploits. For example, focusing on only one weakness in the chain or one composite component might limit the comprehensive understanding of vulnerabilities. RedHat has adopted the practice of chaining multiple CWEs together for the root cause analysis of security flaws in their products, aiming to go beyond tracking a single root cause, which is the current approach to CWEs in the industry[59, 60].

5.3. Conclusion

This paper presents a new PE malware dataset that leverages the use of *Malvuln* to open new doors in malware vulnerability research. Our dataset introduces information unavailable in the other complementary PE malware datasets, such as the presence and annotation of vulnerabilities, obfuscation techniques or ATT&CK. Our contribution brings together the vulnerabilities found in malware from the *Malvuln* dataset and vulnerabilities in benign software from the *ExploitDB* database, *CWE* mapping in both vulnerable classes, *Mitre ATT&CK* Tactics and Techniques from *VulDB*, and analysis of the use of obfuscation across all classes. Additionally, our dataset utilizes Ember to extract the static PE features from all compatible samples to form feature vectors across the four classes to be used for Machine Learning. We obtained benchmark and baseline results on binary malware and benign classifiers, vulnerability detection and multi-class (vulnerable/non-vulnerable, malware/benign) classifiers using these features. From our results, we can derive various assumptions. For all tasks, it is clear that the classifiers could effectively and accurately distinguish between the two cases; *1. malware and benign software*, and also *2. vulnerable and non-vulnerable samples* but to a lesser degree. We also provide an in-depth analysis of the threat and vulnerability mapping and the correlation of vulnerabilities between malware and benign software. By providing these insights and a deep analysis of the importance of exploitable malware vulnerabilities and the potential of their identification as a defensive mechanism, we hope that our contributions encourage further studies into exploitability for defense.

Acknowledgments

We wish to acknowledge funding from the UK Government through the New Deal for Northern Ireland. The funding is delivered on behalf of the Northern Ireland Office and the Department for Science, Innovation and Technology by Innovate UK. Domhnall Carlin is funded by the UKRI EPSRC Research Software Engineering Fellowship (EP/V052284/1).

References

[1] A. Elhadi, M. Maarof, A. Hamza Osman, Malware detection based on hybrid signature behaviour application programming interface call graph, American Journal of Applied Sciences 9 (2012) 283–288.

[2] R. Wang, D.-G. Feng, Y. Yang, P.-R. Su, Semantics-based malware behavior signature extraction and detection method, Journal of Software 23 (2012) 378–393. doi:10.3724/SP.J.1001.2012.03953.

[3] A. Jalilian, Z. Narimani, E. Ansari, Static signature-based malware detection using opcode and binary information, in: Data Science: From Research to Application, Springer, 2020, pp. 24–35. doi:10.1007/978-3-030-37309-2_3.

[4] L. Yang, A. Ciptadi, I. Laziuk, A. Ahmadzadeh, G. Wang, Bodmas: An open dataset for learning based temporal analysis of pe malware, in: 2021 IEEE Security and Privacy Workshops (SPW), 2021, pp. 78–84. doi:10.1109/SPW53761.2021.00020.

[5] H. S. Anderson, P. Roth, EMBER: an open dataset for training static PE malware machine learning models, CoRR abs/1804.04637 (2018). URL: http://arxiv.org/abs/1804.04637. arXiv:1804.04637.

[6] R. E. Harang, E. M. Rudd, SOREL-20M: A large scale benchmark dataset for malicious PE detection, CoRR abs/2012.07634 (2020). URL: https://arxiv.org/abs/2012.07634. arXiv:2012.07634.

[7] R. Ronen, M. Radu, C. Feuerstein, E. Yom-Tov, M. Ahmadi, Microsoft Malware Classification Challenge, 2018. URL: http://arxiv.org/abs/1802.10135, arXiv:1802.10135 [cs].

[8] M. Henriquez, Bugs in malware creating backdoors for security researchers, 2021. URL: https://www.securitymagazine.com/articles/96348-bugs-in-malware-creating-backdoors-for-security-researchers, date accessed: 07-06-2024.

[9] J. Caballero, P. Poosankam, S. McCamant, D. Babi ć, D. Song, Input generation via decomposition and re-stitching: Finding bugs in malware, in: Proceedings of the 17th ACM conference on Computer and communications security, 2010, pp. 413–425. doi:10.1145/1866307.1866354.

[10] N. Singh, U. Pratap Singh, VB2021 paper: Bugs in malware – uncovering vulnerabilities found in malware payloads, Virus Bulletin (2021) 1–14. URL: https://vblocalhost.com/uploads/VB2021-Singh-Singh.pdf.

[11] A. Anubhav, Crash and Burn :: How to crash a Mirai C2 server & why it works., 2019. URL: https://www.ankitanubhav.info/post/crash.

[12] L. Abrams, Vulnerabilities allow hijacking of most ransomware to prevent file encryption, 2022. URL: https://www.bleepingcomputer.com/news/security/lockbit-30-introduces-the-first-ransomware-bug-bounty-program/, date accessed: 15-06-2024.

[13] J. Page, Ransomlord anti-ransomware exploit tool., 2024. URL: https://github.com/malvuln/RansomLord, date accessed: 15-06-2024.

[14] I. Ilascu, Conti, revil, lockbit ransomware bugs exploited to block encryption, 2022. URL: https://web.archive.org/web/20220601204439/https://www.bleepingcomputer.com/news/security/conti-revil-lockbit-ransomware-bugs-exploited-to-block-encryption/, date accessed: 15-06-2024.

[15] E. Kovacs, Vulnerabilities allow hijacking of most ransomware to prevent file encryption, 2022. URL: https://web.archive.org/web/20220504180432/https://www.securityweek.com/vulnerabilities-allow-hijacking-most-ransomware-prevent-file-encryption/, date accessed: 15-06-2024.

[16] A. Calleja, J. Tapiador, J. Caballero, The malsource dataset: Quantifying complexity and code reuse in malware development, IEEE Transactions on Information Forensics and Security 14 (2018) 3175–3190. doi:10.1109/TIFS.2018.2885512.

[17] J. Rosenberg, C. Beek, Examining code reuse reveals undiscovered links among north korea's malware families, 2018. URL: https://www.mcafee.com/blogs/other-blogs/mcafee-labs/examining-code-reuse-reveals-undiscovered-links-among-north-koreas-malware-families/, date Accessed: 09-07-2024.

[18] MalwareTech, Finding the kill switch to stop the spread of ransomware, 2017. URL: https://www.ncsc.gov.uk/blog-post/finding-kill-switch-stop-spread-ransomware-0, date Accessed: 07-06-2024.
[19] J. Fuller, R. P. Kasturi, A. Sikder, H. Xu, B. Arik, V. Verma, E. Asdar, B. Saltaformaggio, C3po: large-scale study of covert monitoring of c&c servers via over-permissioned protocol infiltration, in: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security, 2021, pp. 3352–3365. doi:10.1145/3460120.3484537.
[20] SecurityScorecard, When hackers get hacked: A cybersecurity triumph, 2023. URL: https://app.daily.dev/posts/HpFtuXxKC, date Accessed: 07-06-2024.
[21] A. Gdanski, L. Kessem, From thanos to prometheus: When ransomware encryption goes wrong, 2021. URL: https://securityintelligence.com/posts/ransomware-encryption-goes-wrong/, date accessed: 07-06-2024.
[22] P. Arntz, Oops! black basta ransomware flubs encryption, 2024. URL: https://www.threatdown.com/blog/oops-black-basta-ransomware-flubs-encryption/, date accessed: 07-06-2024.
[23] A. Anubhav, Several iot botnet c2s compromised by a threat actor due to weak credentials., 2019. URL: https://www.ankitanubhav.info/post/c2bruting, date Accessed: 07-06-2024.
[24] S. Walla, C. Rossow, Malpity: Automatic identification and exploitation of tarpit vulnerabilities in malware, in: 2019 IEEE European Symposium on Security and Privacy (EuroS&P), IEEE, 2019, pp. 590–605. doi:10.1109/EuroSP.2019.00049.
[25] H. Griffioen, C. Doerr, Could you clean up the internet with a pit of tar? investigating tarpit feasibility on internet worms, in: 2023 IEEE Symposium on Security and Privacy (SP), IEEE, 2023, pp. 2551–2565. doi:10.1109/SP46215.2023.10179467.
[26] G. Apruzzese, P. Laskov, E. Montes de Oca, W. Mallouli, L. Brdalo Rapa, A. V. Grammatopoulos, F. Di Franco, The role of machine learning in cybersecurity, Digital Threats: Research and Practice 4 (2023) 1–38. doi:10.1145/3545574.
[27] J. Singh, J. Singh, A survey on machine learning-based malware detection in executable files, Journal of Systems Architecture 112 (2021) 101861. URL: https://www.sciencedirect.com/science/article/pii/S1383762120301442. doi:10.1016/j.sysarc.2020.101861.
[28] D. Ucci, L. Aniello, R. Baldoni, Survey of machine learning techniques for malware analysis, Computers & Security 81 (2019) 123–147. doi:10.1016/j.cose.2018.11.001.
[29] R. Sihwail, K. Omar, K. Z. Ariffin, A survey on malware analysis techniques: Static, dynamic, hybrid and memory analysis, Int. J. Adv. Sci. Eng. Inf. Technol 8 (2018) 1662–1671. doi:10.18517/ijaseit.8.4-2.6827.
[30] H. Aghakhani, F. Gritti, F. Mecca, M. Lindorfer, S. Ortolani, D. Balzarotti, G. Vigna, C. Kruegel, When malware is packin'heat; limits of machine learning classifiers based on static analysis features, in: Network and Distributed Systems Security (NDSS) Symposium 2020, 2020.
[31] R. Sihwail, K. Omar, K. A. Zainol Ariffin, S. Al Afghani, Malware Detection Approach Based on Artifacts in Memory Image and Dynamic Analysis, Applied Sciences 9 (2019) 3680. URL: https://www.mdpi.com/2076-3417/9/18/3680. doi:10.3390/app9183680, number: 18 Publisher: Multidisciplinary Digital Publishing Institute.
[32] K. A. Roundy, B. P. Miller, Hybrid analysis and control of malware, in: Recent Advances in Intrusion Detection: 13th International Symposium, RAID 2010, Ottawa, Ontario, Canada, September 15-17, 2010. Proceedings 13, Springer, 2010, pp. 317–338. doi:10.1007/978-3-642-15512-3_17.
[33] B. Cheng, J. Ming, J. Fu, G. Peng, T. Chen, X. Zhang, J.-Y. Marion, Towards paving the way for large-scale windows malware analysis: Generic binary unpacking with orders-of-magnitude performance boost, in: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, 2018, pp. 395–411. doi:10.1145/3243734.3243771.
[34] O. Alrawi, M. Ike, M. Pruett, R. P. Kasturi, S. Barua, T. Hirani, B. Hill, B. Saltaformaggio, Forecasting malware capabilities from cyber attack memory images, in: 30th USENIX security symposium (USENIX security 21), 2021, pp. 3523–3540.
[35] D. G. Corlatescu, A. Dinu, M. P. Gaman, P. Sumedrea, Embersim: A large-scale databank for boosting similarity search in malware analysis, Advances in Neural Information Processing Systems 36 (2024).

[36] M. Sebastián, R. Rivera, P. Kotzias, J. Caballero, Avclass: A tool for massive malware labeling, in: Research in Attacks, Intrusions, and Defenses: 19th International Symposium, RAID 2016, Paris, France, September 19-21, 2016, Proceedings 19, Springer, 2016, pp. 230–253. doi:10.1007/978-3-319-45719-2_11.

[37] S. Sebastián, J. Caballero, Avclass2: Massive malware tag extraction from av labels, in: Proceedings of the 36th Annual Computer Security Applications Conference, 2020, pp. 42–53. doi:10.1145/3427228.3427261.

[38] R. Lyda, J. Hamrock, Using entropy analysis to find encrypted and packed malware, IEEE Security & Privacy 5 (2007) 40–45. doi:10.1109/MSP.2007.48.

[39] X. Ugarte-Pedrero, D. Balzarotti, I. Santos, P. G. Bringas, Sok: Deep packer inspection: A longitudinal study of the complexity of run-time packers, in: 2015 IEEE Symposium on Security and Privacy, IEEE, 2015, pp. 659–673. doi:10.1109/SP.2015.46.

[40] T. Muralidharan, A. Cohen, N. Gerson, N. Nissim, File packing from the malware perspective: techniques, analysis approaches, and directions for enhancements, ACM Computing Surveys 55 (2022) 1–45. doi:10.1145/3530810.

[41] T. Avgerinos, S. K. Cha, A. Rebert, E. J. Schwartz, M. Woo, D. Brumley, Automatic exploit generation, Communications of the ACM 57 (2014) 74–84. doi:10.1145/2560217.2560219.

[42] S. Xu, Y. Wang, Bofaeg: Automated stack buffer overflow vulnerability detection and exploit generation based on symbolic execution and dynamic analysis, Security and Communication Networks 2022 (2022) 1251987. doi:10.1155/2022/1251987.

[43] V. A. Padaryan, V. Kaushan, A. Fedotov, Automated exploit generation for stack buffer overflow vulnerabilities, Programming and Computer Software 41 (2015) 373–380. doi:10.1134/S0361768815060055.

[44] S. Park, D. Kim, S. Jana, S. Son, {FUGIO}: Automatic exploit generation for {PHP} object injection vulnerabilities, in: 31st USENIX Security Symposium (USENIX Security 22), 2022, pp. 197–214.

[45] S. Jan, A. Panichella, A. Arcuri, L. Briand, Automatic generation of tests to exploit xml injection vulnerabilities in web applications, IEEE Transactions on Software Engineering 45 (2017) 335–362. doi:10.1109/TSE.2017.2778711.

[46] T. Wang, T. Wei, G. Gu, W. Zou, Taintscope: A checksum-aware directed fuzzing tool for automatic software vulnerability detection, in: 2010 IEEE Symposium on Security and Privacy, IEEE, 2010, pp. 497–512. doi:10.1109/SP.2010.37.

[47] K. Oosthoek, C. Doerr, Sok: Att&ck techniques and trends in windows malware, in: Security and Privacy in Communication Networks: 15th EAI International Conference, SecureComm 2019, Orlando, FL, USA, October 23-25, 2019, Proceedings, Part I 15, Springer, 2019, pp. 406–425. doi:10.1007/978-3-030-37228-6_20.

[48] Picus Security, Picus red report 2024: The top 10 most prevalent mitre att&ck techniques - the rise of hunter-killer malware, 2024. URL: https://www.picussecurity.com/resource/report/picus-red-report-2024.

[49] Centre for Threat Informed Defense, MITRE ENGENUITY., Top att&ck techniques, 2023. URL: https://top-attack-techniques.mitre-engenuity.org/, date accessed: 11-06-2024.

[50] P. Xu, Y. Zhang, C. Eckert, A. Zarras, Hawkeye: cross-platform malware detection with representation learning on graphs, in: Artificial Neural Networks and Machine Learning–ICANN 2021: 30th International Conference on Artificial Neural Networks, Bratislava, Slovakia, September 14–17, 2021, Proceedings, Part III 30, Springer, 2021, pp. 127–138. doi:10.1007/978-3-030-86365-4_11.

[51] A. T. Nguyen, F. Lu, G. L. Munoz, E. Raff, C. Nicholas, J. Holt, Out of distribution data detection using dropout bayesian neural networks, in: Proceedings of the AAAI Conference on Artificial Intelligence, volume 36, 2022, pp. 7877–7885. doi:10.1609/aaai.v36i7.20757.

[52] L. Van der Maaten, G. Hinton, Visualizing data using t-sne., Journal of machine learning research 9 (2008).

[53] S. Chakraborty, R. Krishna, Y. Ding, B. Ray, Deep learning based vulnerability detection: Are we there yet?, IEEE Transactions on Software Engineering 48 (2021) 3280–3296. doi:10.1109/TSE.

2021.3087402.
[54] T. Bilot, N. El Madhoun, K. Al Agha, A. Zouaoui, A survey on malware detection with graph representation learning, ACM Computing Surveys (2023). doi:10.1145/3664649.
[55] Y. Gao, H. Hasegawa, Y. Yamaguchi, H. Shimada, Malware detection by control-flow graph level representation learning with graph isomorphism network, IEEE Access 10 (2022) 111830–111841. doi:10.1109/ACCESS.2022.3215267.
[56] CISA.gov, Reducing the significant risk of known exploited vulnerabilities, 2022. URL: https://www.cisa.gov/known-exploited-vulnerabilities, date accessed: 15-06-2024.
[57] Vulncheck Kev, Coverage criteria, 2024. URL: https://docs.vulncheck.com/community/vulncheck-kev/coverage-criteria, date accessed: 15-06-2024.
[58] MITRE, Chains and composites, 2023. URL: https://cwe.mitre.org/data/reports/chains_and_composites.html, date Accessed: 16-06-2024.
[59] Red Hat Product Security, cwe-toolkit - cwe chaining concept and tools, 2020. URL: https://github.com/RedHatProductSecurity/cwe-toolkit, date Accessed: 16-06-2024.
[60] Red Hat Customer Portal, Cwe compatibility for red hat customer portal, 2024. URL: https://access.redhat.com/articles/cwe_compatibility, date Accessed: 16-06-2024.
[61] K. Allix, T. F. Bissyandé, J. Klein, Y. Le Traon, Are your training datasets yet relevant? an investigation into the importance of timeline in machine learning-based malware detection, in: International Symposium on Engineering Secure Software and Systems, Springer, 2015, pp. 51–67. doi:10.1007/978-3-319-15618-7_5.
[62] F. Pendlebury, F. Pierazzi, R. Jordaney, J. Kinder, L. Cavallaro, {TESSERACT}: Eliminating experimental bias in malware classification across space and time, in: 28th USENIX security symposium (USENIX Security 19), 2019, pp. 729–746.
[63] A. Guerra-Manzanares, M. Luckner, H. Bahsi, Concept drift and cross-device behavior: Challenges and implications for effective android malware detection, Computers & Security 120 (2022) 102757. doi:10.1016/j.cose.2022.102757.
[64] G. Chin, Correlation vs SHAP: Understanding Feature Importance in ML Models, 2024. URL: https://medium.com/@gawainchin/correlation-vs-shap-understanding-feature-importance-in-ml-models-d6b52b1fba28.
[65] K. Främling, Feature Importance versus Feature Influence and What It Signifies for Explainable AI, 2023. URL: http://arxiv.org/abs/2308.03589, arXiv:2308.03589 [cs].
[66] R. Boemer, Selecting Features With Shapley Values, 2023. URL: https://medium.com/the-ml-practitioner/selecting-features-with-shapley-values-b2da08b5b14c.
[67] R. Kübler, Shapley Values Clearly Explained, 2024. URL: https://towardsdatascience.com/shapley-values-clearly-explained-a7f7ef22b104.

A. Appendix

A.1. Timestamps

It's important to consider the timeline of a dataset in machine-learning-based malware detection and classification systems. This helps in capturing temporal dependencies within the data, thus avoiding experimental bias or incorrect conclusions [61, 62]. Benchmark datasets such as BODMAS, EMBER, and SOREL-20M release their datasets with timestamped samples. For instance, BODMAS uses the first-seen time of a sample based on *VirusTotal* reports, EMBER uses the timestamp in the header from the *COFF* header to split its dataset, and SOREL-20M curates the *first* and *last-seen* times of the samples, eventually using the *first-seen* time for temporal splitting of the data. Similar to BODMAS and SOREL-20M, our dataset is annotated using the *first seen* time from VirusTotal reports, as it has been demonstrated to be robust to alteration, reliable, and accurate compared to other timestamp metrics[63]. The timeseries graphs in Figures 4, 5, 6, and 7 are scaled between the years 2000 and 2024.

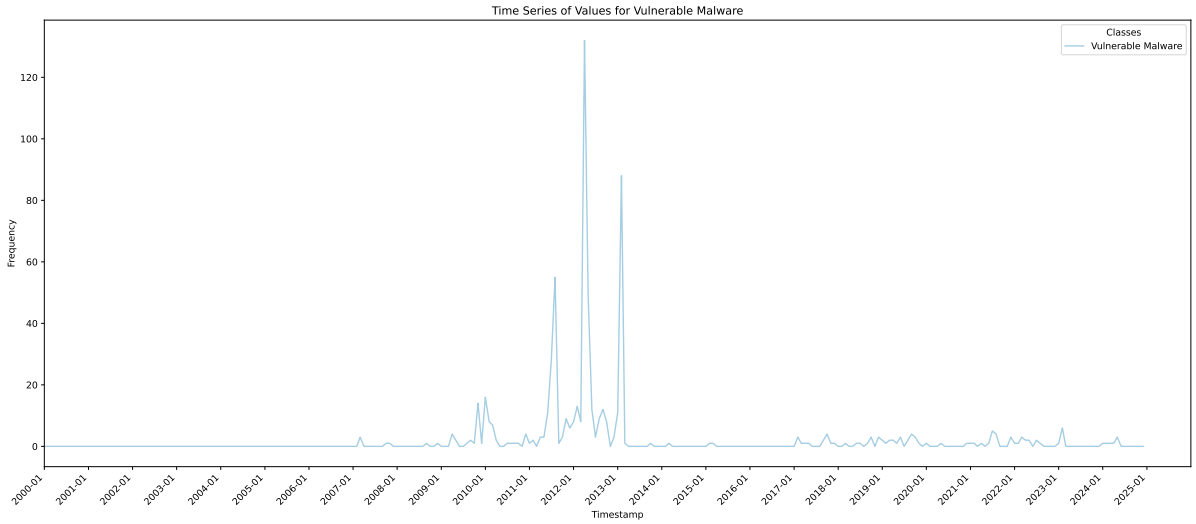

Figure 4: Timeseries graph for Vulnerable Malware from 2000 to 2024.

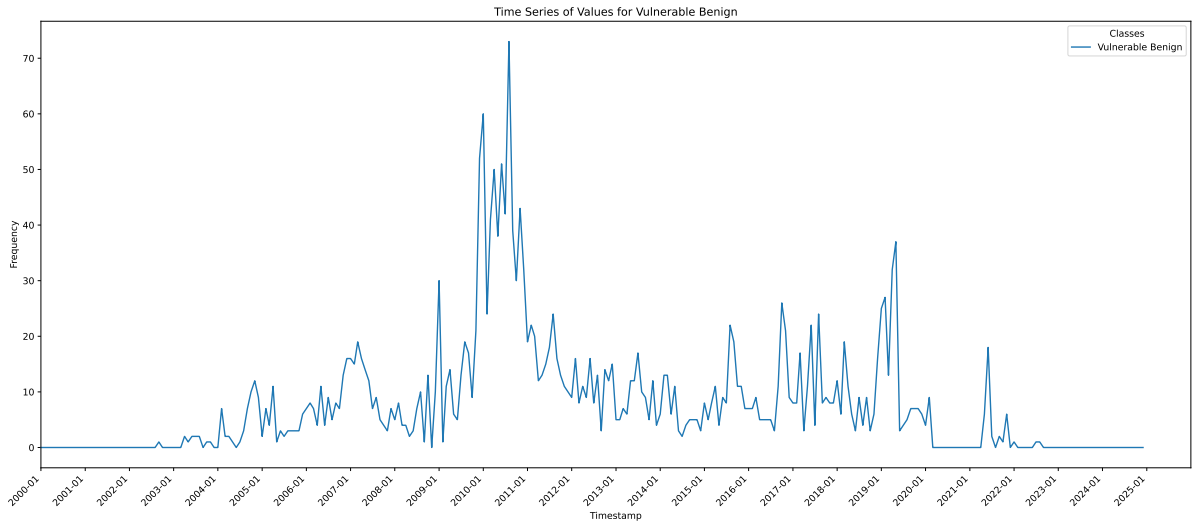

Figure 5: Timeseries graph for Vulnerable Benign from 2000 to 2024.

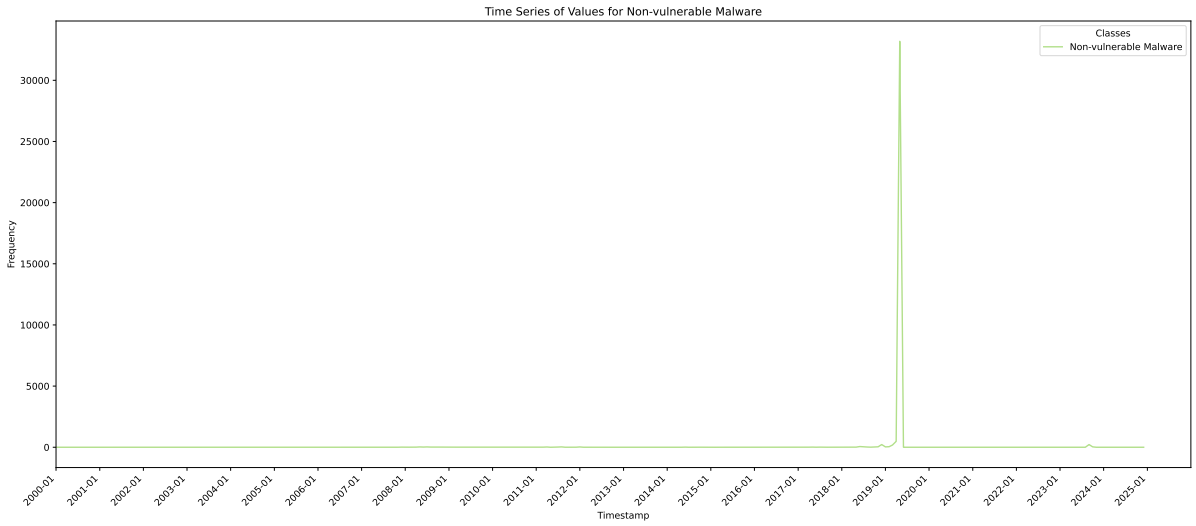

Figure 6: Timeseries graph for Non-vulnerable Malware from 2000 to 2024.

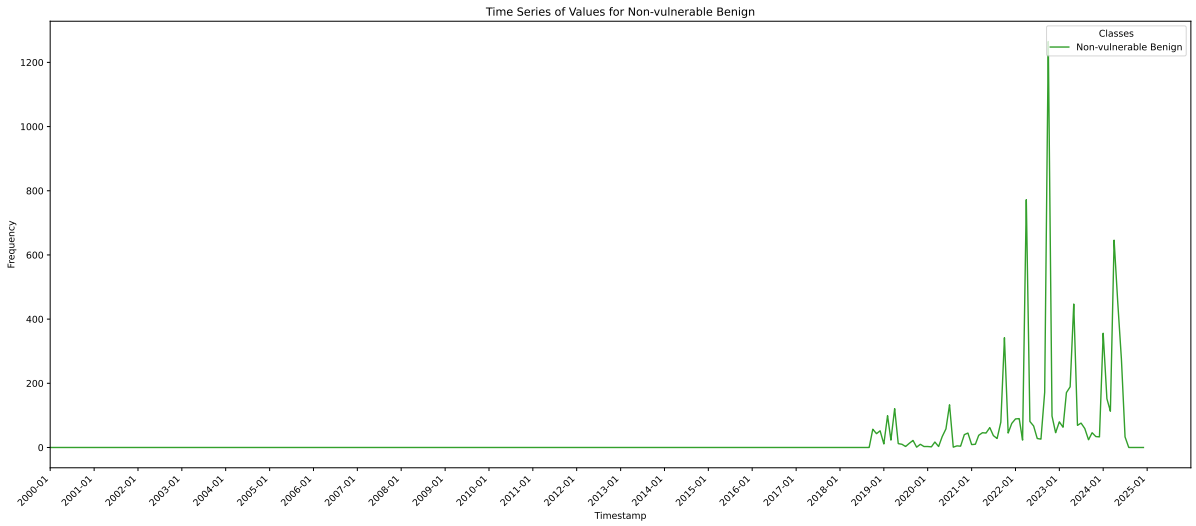

Figure 7: Timeseries graph for Non-vulnerable Benign from 2000 to 2024.

A.2. Binary Classification

A.3. Feature Importance

Feature importance is a concept in machine learning that quantifies the contribution of each feature to the prediction of the model so we can derive explanations for what features influence the model's determinate output. Various studies of use methods such as correlation analysis [64] and Shapley values [65, 66, 67] to assess feature importance. Correlation measures the strength and direction of a linear relationship on a scale from -1 to 1 between features and the target variable, offering a straightforward but overly simplistic view. Whereas Shapley values, which are built from cooperative game theory as a framework explaining the output of a model in correlation to its input features provide a more nuanced and comprehensive evaluation by considering all possible combinations of features and their interactions, which are only bounded by the output magnitude range of the model. This ensures that there is a fair distribution of importance among features which can not only identify key predictive features but also aid in the interpretation of the model and its results. We focused on using Shapley when working with feature importance for our classes, as it also has many methods for plotting graphs

Table 12
Benchmarks on different models for a binary classification task. The classes used were vulnerable malware and non-vulnerable malware

Technique	Model	Accuracy	F1-Score	Precision	Recall
Default Parameters	LGBM	0.943	0.934	0.940	0.943
	RF	0.941	0.929	0.943	0.941
	KNN	0.921	0.912	0.909	0.921
	SVM	0.907	0.865	0.885	0.910
	ANN	0.920	0.918	0.916	0.920
Hyperparameter Optimization	LGBM	0.945	0.938	0.942	0.945
	RF	0.940	0.930	0.939	0.940
	KNN	0.931	0.924	0.923	0.931
	SVM	0.934	0.929	0.927	0.934
	ANN	0.929	0.923	0.921	0.929
Technique	Model	Balanced Accuracy	F1-Score	Precision	Recall
Stratified K-Fold	LGBM	0.754	0.942	0.946	0.948
	RF	0.699	0.929	0.939	0.940
	KNN	0.718	0.923	0.922	0.930
	SVM	0.770	0.938	0.938	0.943
	ANN	0.725	0.928	0.932	0.935

Table 13
Benchmarks on different models for a binary classification task. The classes used were vulnerable malware and vulnerable benign

Technique	Model	Accuracy	F1-Score	Precision	Recall
Default Parameters	LGBM	0.944	0.944	0.944	0.944
	RF	0.934	0.933	0.934	0.934
	KNN	0.929	0.929	0.928	0.929
	SVM	0.909	0.907	0.908	0.909
	ANN	0.922	0.920	0.922	0.922
Hyperparameter Optimization	LGBM	0.932	0.931	0.931	0.932
	RF	0.942	0.941	0.941	0.942
	KNN	0.934	0.934	0.934	0.934
	SVM	0.937	0.936	0.937	0.937
	ANN	0.922	0.920	0.921	0.922
Technique	Model	Balanced Accuracy	F1-Score	Precision	Recall
Stratified K-Fold	LGBM	0.913	0.929	0.930	0.929
	RF	0.910	0.926	0.927	0.926
	KNN	0.899	0.914	0.916	0.914
	SVM	0.912	0.931	0.931	0.932
	ANN	0.909	0.926	0.927	0.926

in its library to better visualize the data. Typically in Shapley graphs, it introduces a gradient scale from red to blue which corresponds to the raw values of the variables for each instance.

- Red points indicate high-impact SHAP values, suggesting that the feature value led the model to increase its prediction.
- Blue points indicate low-impact SHAP values, suggesting that the feature value led the model to decrease its prediction.

Table 14
Benchmarks on different models for a binary classification task. The classes used were vulnerable malware and non-vulnerable benign

Technique	Model	Accuracy	F1-Score	Precision	Recall
Default Parameters	LGBM	0.996	0.996	0.996	0.996
	RF	0.996	0.996	0.996	0.996
	KNN	0.999	0.999	0.999	0.999
	SVM	0.995	0.995	0.995	0.995
	ANN	0.995	0.995	0.995	0.995
Hyperparameter Optimization	LGBM	0.997	0.997	0.997	0.997
	RF	0.996	0.996	0.996	0.996
	KNN	0.999	0.999	0.999	0.999
	SVM	0.998	0.998	0.998	0.998
	ANN	0.998	0.998	0.998	0.998
Technique	Model	Balanced Accuracy	F1-Score	Precision	Recall
Stratified K-Fold	LGBM	0.989	0.996	0.996	0.996
	RF	0.967	0.991	0.991	0.991
	KNN	0.983	0.995	0.995	0.995
	SVM	0.990	0.996	0.996	0.996
	ANN	0.990	0.996	0.996	0.996

Table 15
Benchmarks on different models for a binary classification task. The classes used were vulnerable benign and non-vulnerable benign

Technique	Model	Accuracy	F1-Score	Precision	Recall
Default Parameters	LGBM	0.999	0.999	0.999	0.999
	RF	0.996	0.996	0.996	0.996
	KNN	0.999	0.999	0.999	0.999
	SVM	0.997	0.997	0.997	0.997
	ANN	0.999	0.999	0.999	0.999
Hyperparameter Optimization	LGBM	0.999	0.999	0.999	0.999
	RF	0.998	0.998	0.998	0.998
	KNN	0.999	0.999	0.999	0.999
	SVM	0.999	0.999	0.999	0.999
	ANN	0.999	0.999	0.999	0.999
Technique	Model	Balanced Accuracy	F1-Score	Precision	Recall
Stratified K-Fold	LGBM	0.998	0.998	0.998	0.998
	RF	0.993	0.996	0.996	0.996
	KNN	0.995	0.997	0.997	0.997
	SVM	0.995	0.996	0.996	0.996
	ANN	0.995	0.996	0.996	0.996

It is worth noting that SHAP gives insights into the model's behavior in the context of the data used, but causal relationships or generalizable patterns with unseen data could be overlooked. SHAP could also generate unexpected relationships between features and predictions which indicate possible data issues or model artefacts.

From the SHAP results in Figure 8, which are derived from the LGBM model performing multi-class classification on the data, many observations can be made. The first of which we can see is that feature 637 had a high positive impact on both vulnerable and non-vulnerable malware predictions. Whereas

Table 16
Benchmarks on different models for a binary classification task. The classes used were vulnerable benign and non-vulnerable malware

Technique	Model	Accuracy	F1-Score	Precision	Recall
Default Parameters	LGBM	0.971	0.971	0.971	0.971
	RF	0.958	0.958	0.958	0.958
	KNN	0.958	0.958	0.958	0.958
	SVM	0.946	0.945	0.945	0.946
	ANN	0.952	0.953	0.958	0.952
Hyperparameter Optimization	LGBM	0.970	0.970	0.970	0.970
	RF	0.970	0.967	0.967	0.967
	KNN	0.960	0.960	0.960	0.960
	SVM	0.968	0.968	0.969	0.968
	ANN	0.963	0.964	0.964	0.963
Technique	Model	Balanced Accuracy	F1-Score	Precision	Recall
Stratified K-Fold	LGBM	0.953	0.973	0.973	0.973
	RF	0.931	0.962	0.962	0.962
	KNN	0.933	0.961	0.961	0.962
	SVM	0.953	0.968	0.969	0.968
	ANN	0.923	0.956	0.956	0.956

Table 17
Benchmarks on different models for a binary classification task. The classes used were non-vulnerable benign vs non-vulnerable malware

Technique	Model	Accuracy	F1-Score	Precision	Recall
Default Parameters	LGBM	0.998	0.998	0.998	0.998
	RF	0.996	0.996	0.996	0.996
	KNN	0.996	0.996	0.996	0.996
	SVM	0.998	0.998	0.998	0.998
	ANN	0.997	0.997	0.997	0.997
Hyperparameter Optimization	LGBM	0.999	0.999	0.999	0.999
	RF	0.998	0.998	0.998	0.998
	KNN	0.996	0.996	0.996	0.996
	SVM	0.998	0.998	0.998	0.998
	ANN	0.997	0.997	0.997	0.997
Technique	Model	Balanced Accuracy	F1-Score	Precision	Recall
Stratified K-Fold	LGBM	0.997	0.997	0.997	0.997
	RF	0.995	0.995	0.996	0.995
	KNN	0.995	0.995	0.995	0.995
	SVM	0.997	0.997	0.997	0.997
	ANN	0.996	0.996	0.996	0.996

feature 637 had a significantly low positive impact in its prediction for non-vulnerable benign with more weight being in the negative direction. After investigating this feature, we found that it exists in the COFF File Header, specifically the machine type[30]. There were 2 machine types identified; *I386* and *AMD64*. The *I386* machines consistently dominate across all classes compared to the *AMD64*, with 557/3 in VM, 1,408/4 in VB, 32,615/213 in NVM, and 2,910/2222 in NVB. From this distribution, we can see that, while *I386* machines are predominantly more common in VM, VB, and NVM classes, the NVB

[30]https://learn.microsoft.com/en-us/windows/win32/debug/pe-format

Table 18
Benchmarks on different models for a binary classification task. The classes used were vulnerable malware vs all benign samples

Technique	Model	Accuracy	F1-Score	Precision	Recall
Default Parameters	LGBM	0.983	0.983	0.983	0.983
	RF	0.979	0.979	0.979	0.979
	KNN	0.974	0.975	0.975	0.974
	SVM	0.971	0.971	0.971	0.971
	ANN	0.980	0.980	0.981	0.980
Hyperparameter Optimization	LGBM	0.980	0.980	0.981	0.980
	RF	0.979	0.979	0.979	0.979
	KNN	0.977	0.977	0.978	0.977
	SVM	0.982	0.982	0.982	0.982
	ANN	0.981	0.981	0.982	0.981
Technique	Model	Balanced Accuracy	F1-Score	Precision	Recall
Stratified K-Fold	LGBM	0.930	0.937	0.979	0.922
	RF	0.902	0.932	0.974	0.917
	KNN	0.921	0.935	0.977	0.920
	SVM	0.926	0.940	0.978	0.926
	ANN	0.920	0.934	0.978	0.919

Table 19
Benchmarks on different models for a binary classification task. The classes used were vulnerable benign vs all malware samples

Technique	Model	Accuracy	F1-Score	Precision	Recall
Default Parameters	LGBM	0.967	0.967	0.967	0.967
	RF	0.962	0.961	0.961	0.962
	KNN	0.950	0.950	0.950	0.95
	SVM	0.943	0.943	0.942	0.943
	ANN	0.959	0.958	0.958	0.959
Hyperparameter Optimization	LGBM	0.962	0.962	0.962	0.962
	RF	0.962	0.962	0.962	0.962
	KNN	0.954	0.954	0.954	0.954
	SVM	0.960	0.960	0.960	0.960
	ANN	0.950	0.951	0.953	0.950
Technique	Model	Balanced Accuracy	F1-Score	Precision	Recall
Stratified K-Fold	LGBM	0.940	0.963	0.964	0.964
	RF	0.924	0.957	0.957	0.958
	KNN	0.920	0.953	0.954	0.954
	SVM	0.941	0.959	0.960	0.958
	ANN	0.924	0.954	0.955	0.954

class has a relatively higher proportion of *AMD64* machines, which explains the data distribution in Figure 2. A broader comparison was made in Table 20 by demonstrating the distribution of features across the top 15 SHAP values for each class.

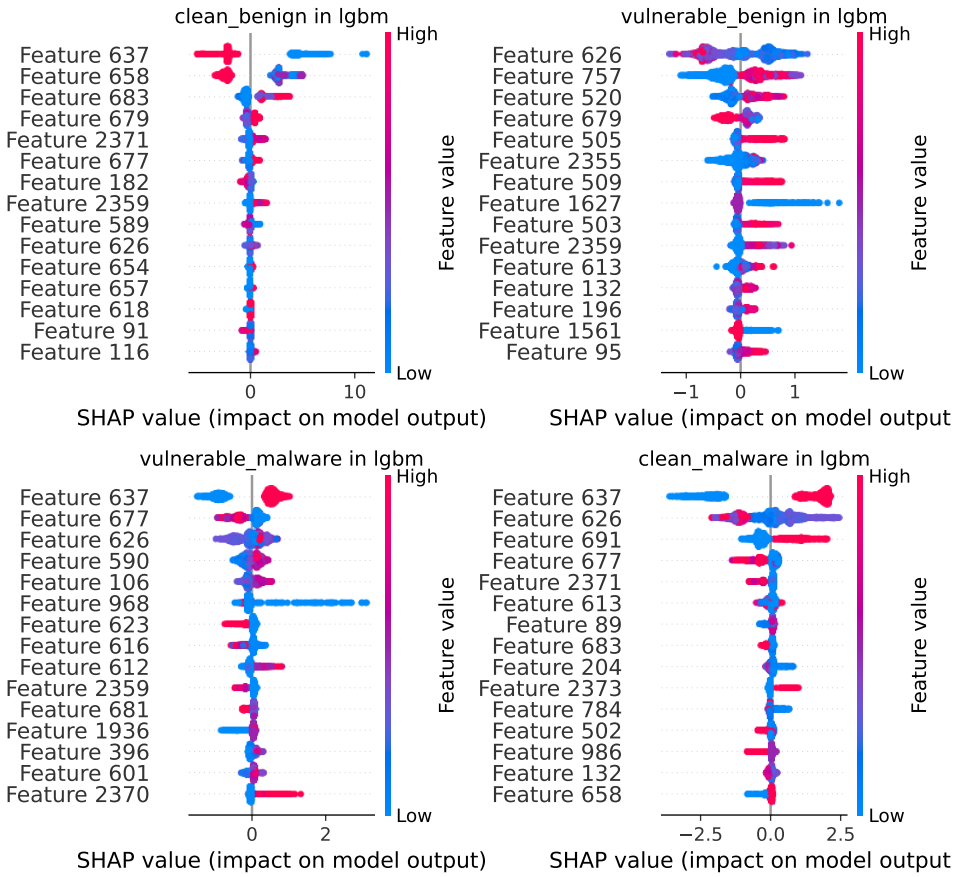

Figure 8: SHAP Summary Plot of Feature Contributions for the Multi-Class Classification Task in LGBM Model

Table 20
showing the distribution of top 15 SHAP features per class with the corresponding feature description.

Feature	NVB	VB	NVM	VM	Feature Description
89				X	ByteHistogram-88
91	X				ByteHistogram-90
95		X			ByteHistogram-94
106			X		ByteHistogram-105
116	X				ByteHistogram-115
132		X		X	ByteHistogram-131
182	X				ByteHistogram-181
196		X			ByteHistogram-195
204				X	ByteHistogram-203
396			X		ByteEntropyHistogram-139
502				X	ByteEntropyHistogram-245
503		X			ByteEntropyHistogram-246
505		X			ByteEntropyHistogram-248
509		X			ByteEntropyHistogram-252
520		X			frequency of printable characters - $
589	X				frequency of printable characters - i
590			X		frequency of printable characters - j
601			X		frequency of printable characters - u
612			X		entropy of the byte histogram
613		X		X	occurrences of the string "c:\" (ignore case)
616			X		occurrences of "MZ"
618	X				virtual size of the lief parsed binary
623			X		whether the binary has a "Resources" object
626	X	X	X	X	# of Symbols
637	X		X	X	hash COFF machine type - 9
654	X				hash optional subsystem - 6
657	X				hash optional subsystem - 9
658	X			X	hash optional dll_characteristics - 0
677	X		X	X	hash optional magic number - 9
679	X	X			minor_image_version
681			X		minor_linker_version
683	X			X	minor_operating_system_version
691				X	# of sections with empty name
757		X			hash on pair of section name and entropy - 13
784				X	hash on pair of section name and entropy - 40
968			X		hash on list of unique imported libraries - 24
986				X	hash on list of unique imported libraries - 42
1561		X			hash on list of library:function - 361
1627		X			hash on list of library:function - 427
1936			X		hash on list of library:function - 736
2355		X			virtual size of data directories - 1
2359	X	X	X		virtual size of data directories - 3
2370			X		size of data directories - 9
2371	X			X	virtual size of data directories - 9
2373				X	virtual size of data directories - 10