A Machine Learning Framework for Malware Triage

by

Soroush Danaeifard

A Project Submitted in Partial Fulfillment of the Requirements for the Degree of

MASTER OF ENGINEERING

In the Department of Electrical and Computer Engineering

University of Victoria

Supervisory Committee

A Machine Learning Framework for Malware Triage

by

Soroush Danaeifard

University of Victoria 2024

Supervisory Committee
Dr. Issa Traore, Supervisor, Department of Electrical and Computer Engineering
Dr. Isaac Woungang, Co-Supervisor, Department of Electrical and Computer Engineering

List of Contents

List of Figures

List of Tables

Glossary

PE: Portable Executable

AI: Artificial Intelligence

ML: Machine Learning

Malware: Malicious Software

IDS/IPS: Intrusion Detection/Prevention Systems

SIEM: Security Information and Event Management

EPP: Endpoint Protection Platform

EDR: Endpoint Detection and Response

IOC: Indicators of Compromise

JSON: JavaScript Object Notation

NLP: Natural Language Processing

POS: Part of Speech

NER: Named Entity Recognition

Abstract

Every day, thousands of new malicious software emerge globally, posing threats to consumer devices, stealing private data, or inducing financial losses. The increasing number and sophistication of malware threats underscores the need for effective and efficient malware detection and triage schemes.

Malware triage is a process used by cybersecurity professionals to quickly assess, prioritize, and respond to malware incidents. Effective malware triage requires a combination of automated tools, skilled personnel, and well-defined procedures to quickly and accurately respond to malware incidents, minimizing damage and recovery time.

An essential aspect of the triage is the categorization and prioritization task which enables determining malware type. Automating such process helps save valuable time for cybersecurity analysts and investigators.

This report explores the application of machine learning techniques as a secondary layer for assigning systematically a category after a file has been flagged as malicious by a detector. Three different classification techniques were studied: Naive Bayes, Random Forest, and K-nearest Neighbors. Experimental evaluation on a public dataset yielded excellent performance for Random Forest compared to the other two classifiers.

Acknowledgments

I want to thank my supervisor, **Dr. Issa Traore**, for its constant guidance, support, and motivation throughout my project and throughout my program.

I would like to thank my co-supervisor, **Dr. Isaac Woungang**, for all his valuable advice and feedback.

Dedication

I would like to dedicate this report to my son and my wife, Maryam, for their consistent support and encouragement during my academic pursuits. Additionally, I extend this dedication to my dear family and close friends, whose unwavering inspiration has been a guiding light throughout this journey.

Chapter 1: Introduction

1.1 Context

In the interconnected world of today, where digital communication is everywhere, the danger of malware is a major concern for all networks. Malware, which stands for malicious software, presents a significant threat to the security and reliability of networks worldwide. From secret data theft to disruptive denial-of-service attacks, malware takes on different forms, each capable of causing chaos to systems and exposing sensitive information.

The core of network malware detection is its capacity to spot and stop these sneaky threats before they cause damage. By closely examining the network traffic and studying the behavior patterns, detection systems can identify unusual activities that suggest malicious intent. This allows for quick and effective actions to be taken in response.

1.2 The Growing threat of Malware

As there is an increase in internet-connected devices and hackers develop more sophisticated techniques, the danger posed by cyber threats has severely grown (Figure 1.1). Malware developers use advanced methods like polymorphism and obfuscation to avoid the detection by traditional security measures, making it difficult to stop their harmful activities. The results of a successful malware attack can be devastating, leading to financial losses, damage to reputation, legal consequences, and threats to national security [1].

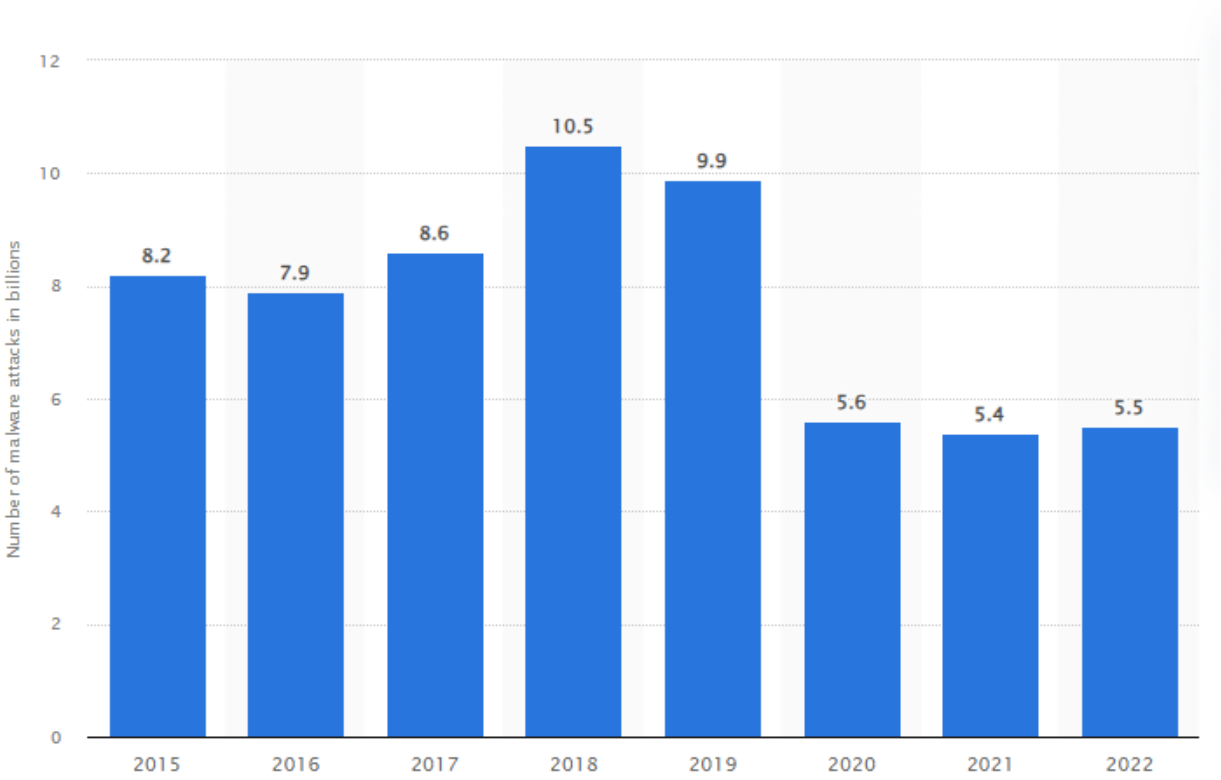

Figure 1.1 Annual number of malware attacks worldwide from 2015 to 2022 [1].

1.3. The Imperative of Detection

Considering the aforementioned increasing dangers, the importance of robust malware detection mechanisms cannot be overstated. Network administrators and cybersecurity professionals rely on these systems to safeguard the critical infrastructure, protect sensitive data, and preserve the trust of users [2]. Effective detection not only mitigates the immediate impact of malware attacks but also serves as a deterrent against future incursions, thus promoting a safer and stronger network environment.

1.4 Challenges and Opportunities

Detecting malware in network traffic poses a significant challenge, fraught with complexities and uncertainties. The huge amount of data moving through today's networks requires scalable and effective detection algorithms that can instruct on the difference between harmless and harmful activities instantly. Also, the rise of encrypted communication protocols makes it hard for traditional detection methods to work well because malicious content can hide within the encrypted data.

Nevertheless, among these challenges lie the opportunities for innovation and advancement. New technologies like artificial intelligence (AI), machine learning (ML), and behavioral analytics show potential for improving how well malware detection systems work. By using these advanced tools, researchers and professionals can create defense systems that can adapt and withstand the changing malware threats, staying ahead of them.

1.5. Objectives of Research

The primary objective of this research is to explore and evaluate a ML-based approach for malware categorization. Malware categorization is essential for security analysts who are responsible for triage activities. After a file has been flagged as malware, knowing the exact category or type helps focusing on the incident response task.

The proposed model:

- acts as a secondary layer after a detector has flagged a file as malware.
- utilizes ML models to classify the malware.

Figure 1.2 outlines a typical detection architecture and how the proposed model fits in it:

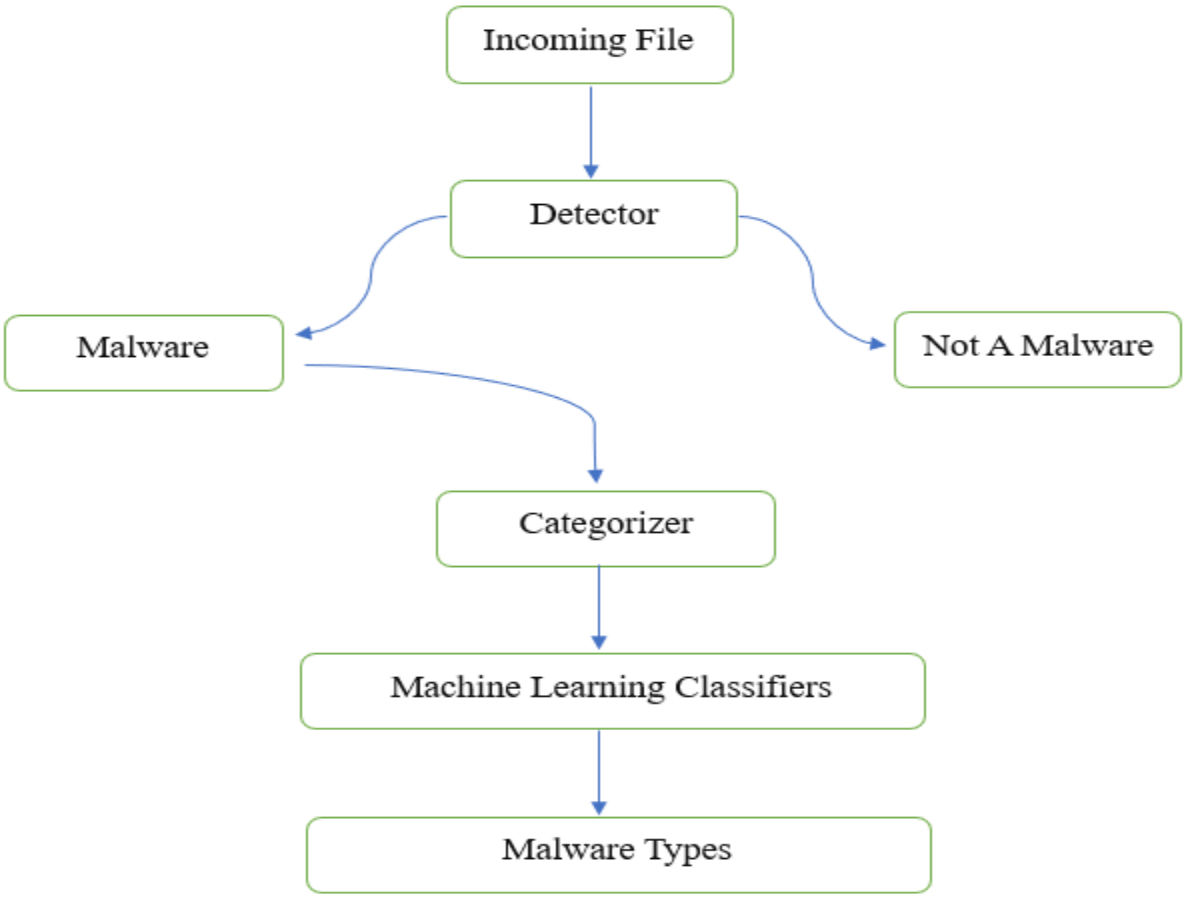

Figure 1.2 A typical detection architecture

The proposed approach leverages the natural language processing (NLP) techniques for data processing and feeds the outcome to a ML classifier. Three different classifiers are studied and evaluated using an existing public dataset.

The remaining chapters are organized as follows:

- Chapter 2 provides some background information on malware analysis and classification.
- Chapter 3 presents the dataset used to evaluate our proposed approach.
- Chapter 4 presents the proposed approach and its evaluation using the three different classification techniques.
- Chapter 5 makes some concluding remarks and highlights some future works in the scope of this project.

Chapter 2: Background

Network malware detection is a critical aspect of cybersecurity that aimed at identifying and mitigating malicious software (i.e. malware) threats within computer networks. Malware encompasses a wide range of malicious software programs designed to disrupt, damage, or gain unauthorized access to computer systems and data. Network malware detection involves the use of various techniques and technologies to identify and prevent the spread of malware within networked environments.

2.1 Context

The expansion of malware poses significant risks to organizations, individuals, and critical infrastructure systems. Malware can be distributed through various vectors, including email attachments, malicious websites, removable media, and network vulnerabilities [3]. Once inside a network, malware can propagate rapidly, causing a widespread damage, as well as data breaches, financial losses, to name a few.

Traditional signature-based detection methods, which rely on predefined patterns or signatures of known malware, have limitations in detecting new and evolving threats. As a result, there has been a shift towards more advanced detection approaches that leverage the behavioral analysis and anomaly detection to identify previously unseen malware variants and zero-day attacks [4].

In terms of collecting data, there are two specific ways:

- **Statistical Methods**: This involves collecting and analyzing data from existing sources, such as surveys, experiments, observational studies, or datasets that have been pre-compiled. The data is often structured and labeled, making it suitable for training supervise learning model.
- **Simulations or Synthetic Data Generation:** This involves creating artificial data that mimics real-world scenarios. This can be done through sandbox environments, where models are trained on simulated scenarios, or by generating synthetic data using algorithms that can replicate the patterns and distributions found in real data. This approach is useful when real data is scarce, sensitive, or difficult to obtain.

When applying these data-gathering methods to network detection, the approaches are tailored to the specific needs and challenges of monitoring and identifying anomalies or threats in a network.

Here is how each approach can be expanded:

1. Statistical Methods in Network Detection:

- **Data Collection from Network Traffic:** This involves collecting data from various points in a network, such as routers, firewalls, and endpoints. The data includes logs, packet captures, and flow records. Statistical methods can be used to analyze this data to identify patterns, trends, and anomalies.
- **Baseline and Anomaly Detection:** By using statistical techniques, a baseline of normal network behavior is established. Any deviations from this baseline, such as unusual traffic patterns or spikes in data transfers, can be flagged as potential security threats.

- **Feature Engineering:** Specific features are extracted from the network traffic data, such as connection duration, packet size, and protocol type. Statistical analysis helps in selecting and refining these features to improve the accuracy of detection models.
- **Correlation Analysis:** Statistical methods can be used to correlate different events and logs across the network, helping to identify coordinated attacks or complex threat patterns.

2. Simulations or Synthetic Data Generation in Network Detection:

- **Sandboxing for Threat Detection:** In network detection, sandboxing refers to isolating potentially malicious files, code, or traffic in a controlled environment to observe their behavior. The sandbox generates data on how these threats interact with the network, which can be used to train detection model.
- **Simulation of Network Attacks:** Simulated attacks, such as Distributed Denial of Service (DDoS) or phishing attempts, are run in a controlled environment to generate data that mimics real-world scenarios. This data can then be used to train machine learning models to detect such attacks in actual networks.
- **Synthetic Traffic Generation:** When real network traffic data is not available or is insufficient, synthetic traffic can be generated to simulate various network conditions and attack scenarios. This synthetic data is used to train and test network detection models.
- **Adversarial Training:** In some cases, synthetic data is generated to simulate adversarial attacks, where the goal is to fool the detection system. This data helps in training more robust models that can withstand sophisticated evasion techniques.

2.2 Approaches to Network Malware Detection

- **Signature-Based Detection:** This approach involves comparing the network traffic or file contents against a database of known malware signatures. While effective against known threats, signature-based detection is limited in its ability to detect novel or polymorphic malware variants.
- **Heuristic Analysis:** This approach involves identifying the suspicious patterns or behaviors in the network traffic or file activities that may indicate the presence of malware. This approach also relies on predefined rules or heuristics to detect potentially malicious behavior.
- **Behavior-based Detection:** This approach focuses on analyzing the actions and activities of software programs to identify any anomalous behavior that is indicative of a malware. This approach also monitors the system processes, network communications, and file interactions to detect the deviations from the normal behavior.
- **ML and AI:** ML techniques, such as supervised learning, unsupervised learning, and deep learning, are increasingly being used for network malware detection [5]. These approaches are meant to analyze large volumes of network data, in order to identify the patterns and anomalies that are associated with the malware activity.
- **Anomaly Detection:** Anomaly detection techniques aim to identify the deviations from the normal network behavior, that may indicate the presence of malware. This approach also involves establishing a baseline of normal network activity and flagging any deviations from that baseline as potential threats.

- **Sandboxing and Emulation:** Sandboxing involves executing suspicious files or programs in isolated environments to observe their behavior and determine if they exhibit malicious activities [6]. Besides, emulation techniques simulate the execution of malware in controlled environments to analyze their behavior without risking the integrity of the production systems.
- **Integrated Security Solutions:** Many modern network security solutions integrate multiple detection techniques, such as intrusion detection/prevention systems (IDSs/IPSs), firewalls, endpoint protection platforms (EPP), and security information and event management (SIEM) systems, to provide a comprehensive protection against malware threats.

Figure 2.1 depicts the percentage usage of various detection techniques.

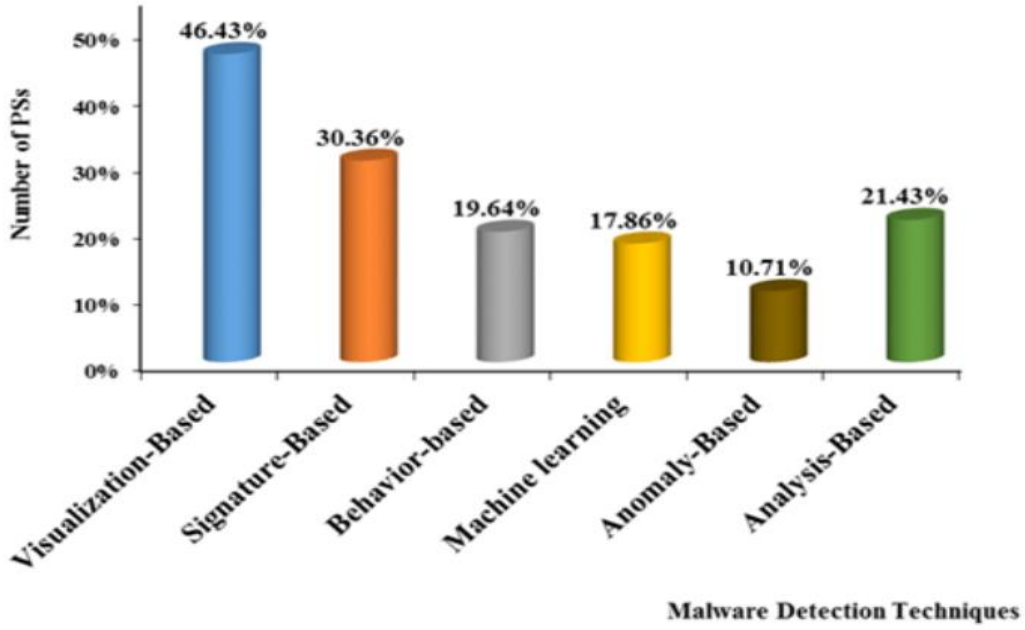

Figure 2. 1 Malware Detection Techniques Usage Percentage [7]

Network malware detection is a multifaceted field that requires a combination of techniques and approaches to effectively identify and mitigate the malware threats in networked environments. As malware continues to evolve and become more sophisticated, it is essential for organizations to deploy robust detection mechanisms and stay vigilant against emerging threats. Continuous research and development in network malware detection are vital to staying ahead of evolving cyber threats and protecting the critical assets from malicious actors.

2.3 What is Malware

Malware, short for "malicious software", is any software intentionally designed to cause damage to a computer, server, network, or device, or to gain unauthorized access to sensitive information. Malware encompasses a wide range of malicious programs or code, including viruses, worms, Trojans, ransomware, spyware, adware, rootkits, to name a few.

Malware can disrupt the normal computing operations, steal the personal or confidential data, compromise the system security, and propagate itself to other systems [8]. It often relies on social engineering tactics, software vulnerabilities, or other means, to infiltrate and infect the target systems. Malware poses significant threats to individuals, organizations, and society at large, requiring proactive measures such as antivirus software, firewalls, security patches, and user education, to mitigate its risks.

2.4 How Malware Works

Malware typically infects a system through various means such as email attachments, malicious websites, infected USB drives, or vulnerabilities in software or operating systems [9].

Once the malware gains access to a system, it executes its malicious code. This code can perform a wide range of actions depending on the type and purpose of the malware. Common actions include stealing sensitive information, propagating encrypting files for ransom purpose, damaging the system's files, or turning the infected system into part of a botnet.

Many malware strains attempt to establish a persistence on the infected systems, in order to ensure they remain active, even after reboots or attempts to remove them. They might create registry entries, modify the system files, or use other techniques to achieve persistence. Malware often communicates with the remote servers controlled by the attackers to receive the commands, download additional payloads, or exfiltrate the stolen data [10]. This communication typically occurs over the Internet using protocols such as HTTP, HTTPS, or DNS.

2.5 How to Capture Data

Capturing the data related to malware activities is essential for analysis, detection, and response.

Some common relevant techniques are described as follow:

2.5.1 Network Traffic Analysis

Network traffic can be monitored using tools such as Wireshark or intrusion detection systems (IDSs) to capture the communication between the infected system and the remote servers. The traffic can then be analyzed to identify malicious activities, such as command-and-control communication or data exfiltration.

An overview of data capture through network traffic is presented in Figure 2.2.

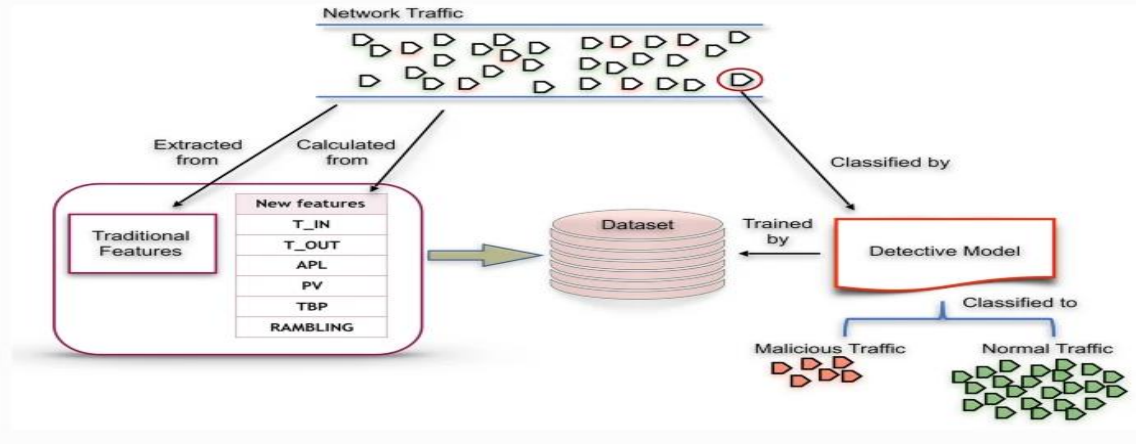

Figure 2. 2 Capturing Data by Network Traffic [11].

2.5.2 System Logs

System logs can be collected and analyzed from various sources such as event logs (Windows Event Viewer), syslog (Unix-based systems), or application logs. Then, suspicious or anomalous activities such as unusual process execution, file modifications, or authentication failures, can be identified and looked over.

2.5.3 Endpoint Detection and Response

End point detection and response (EDR) solutions can be deployed on the endpoints to continuously monitor for malicious behavior and capture the relevant data, such as process activity, file modifications, and network connections. These solutions can provide valuable insights into the malware activities and help with the incident response.

2.5.4 Memory Forensics

The memory of infected systems can be analyzed using tools such as Volatility, to identify the running processes, loaded modules, and other artifacts associated with the malware execution. Memory forensics can uncover the stealthy malware that may not be visible through traditional disk-based analysis.

2.5.5 File Analysis

Suspicious files can be analyzed using malware analysis tools such as sandbox environments or antivirus scanners. These tools can help identifying the behavior and characteristics of the malware, as well as the indicators of compromise (IOCs), which can then be used for detection and prevention purpose.

2.6 JSON With Tree Structure

JSON (JavaScript Object Notation) is a lightweight data interchange format commonly used for representing the structured data. While JSON itself doesn't enforce any specific structure, it is often used to represent hierarchical or tree-like data structures, making it a popular choice for configuration files, data exchange between systems, and the storage of various types of data [12].

Here is how JSON files can be used to represent the tree structures:

- ✓ **Nodes**: In a tree structure, each element or entity is represented as a node. In JSON, each node is typically represented as an object (i.e. a collection of key-value pairs).

- ✓ **Properties**: Nodes in the tree can have properties that describe or characterize them. These properties are represented as key-value pairs within the JSON object. Common properties might include a node's name, value, type, or any other relevant information.
- ✓ **Hierarchy**: One of the defining features of a tree structure is its hierarchy, where the nodes are organized into parent-child relationships. In JSON, this hierarchy is represented using nested objects. A parent node can have one or more child nodes, each represented as a nested object within the parent node's object.
- ✓ **Arrays for Multiple Children**: If a node can have multiple child nodes, they are typically represented as an array within the parent node's object. This allows for flexibility in representing varying numbers of child nodes.
- ✓ **Leaf Nodes vs. Branch Nodes**: In a tree structure, nodes that do not have any child nodes are often referred to as leaf nodes, while nodes that have one or more child nodes are referred to as branch nodes. Leaf nodes are represented as objects without any nested child nodes, while branch nodes have nested objects representing their children.

2.7 Parsing JSON using Natural Language Processing

Parsing JSON with a tree structure using Natural Language Processing (NLP) typically involves extracting the structured information from the textual descriptions or unstructured data [13]. While NLP techniques can help in understanding the content and context of JSON data, directly parsing JSON with a tree structure using NLP alone might not be the most efficient approach.

However, one can use NLP techniques in conjunction with traditional JSON parsing methods to extract the information from the textual descriptions or to assist in understanding the semantics of the data [14].

- **Text Understanding**: NLP techniques such as named entity recognition (NER), part-of-speech tagging (POS), and dependency parsing, can be used to understand the textual descriptions associated with the JSON data. This can help in identifying the key entities, relationships, and actions that are described in the text.
- **Information Extraction**: Relevant information can be extracted from the textual descriptions that describe the structure of the JSON tree. For instance, if the JSON data describes a hierarchical organization, one can use NLP to identify the keywords or phrases indicating the parent-child relationships such as "parent node", "child node", "branch", "leaf", to name a few.
- **Semantic Analysis**: The semantics of the textual descriptions can be analyzed to understand the intended meaning behind the JSON structure. This may involve identifying the synonyms, hypernyms, and hyponyms, to capture the variations in the language used to describe the tree structures.
- **Mapping to JSON Structure**: Once the textual descriptions have been extracted and analyzed, the information can be mapped to the corresponding JSON structure. The insights gained from NLP can then be used to identify the hierarchical relationships between the nodes, the properties of each node, and any other relevant information.
- **JSON Parsing**: Finally, the JSON data based on the mapped structure can be parsed. To this end, traditional JSON parsing techniques can be used, such as libraries or built-in functions provided by the programming language. These parsing techniques can help, converting the JSON data into a structured format, that can then be easily manipulated and processed.

A sample JSON file parsing is depicted in Figure 2.3.

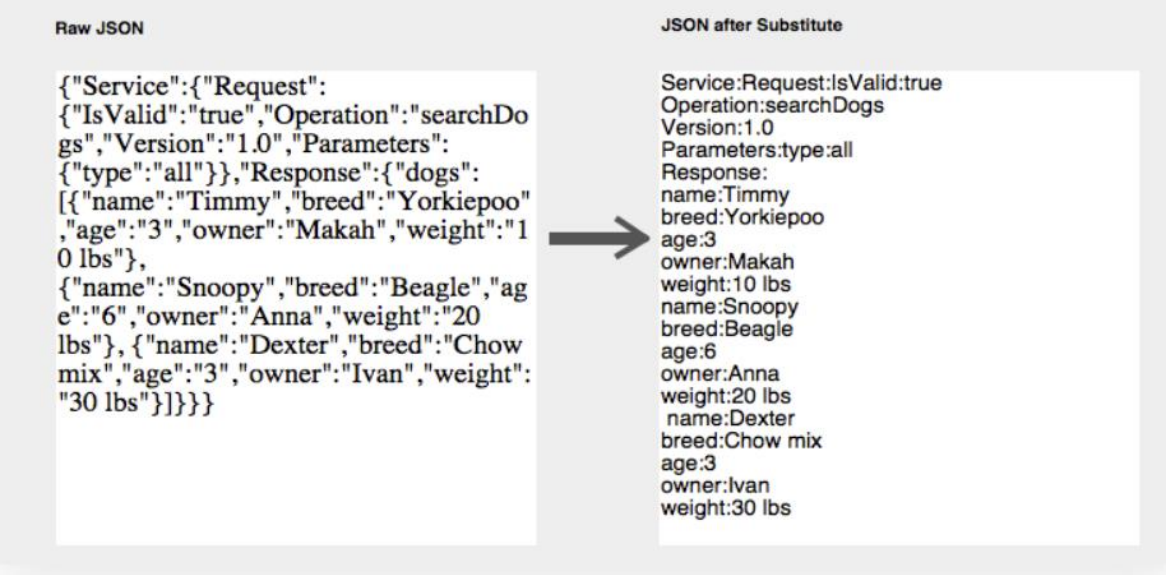

Figure 2. 3 Raw JSON File Parsing Sample [15].

Chapter 3: Dataset for Experiments

For the experiments, we use an existing public malware dataset called CAPEv2.

3.1 CAPEv2 Sandbox

CAPEv2 is an open-source sandbox [16] for execution analysis of samples that started as a fork of the original Cuckoo sandbox in 2019. It is still actively developed and maintained at the time of collecting the reports. It further extends the monitoring and execution capabilities of the Cuckoo sandbox, including unpacking and/or capturing payload that provides more detailed information about the executed samples. It also allows the detection based on the Yara signatures.

The actual execution of the samples and collection of the reports were performed at the Czech Technical University, Prague, in July and August 2021, they used the recent version of CAPEv2 sandbox at that time [17]. They ran the collection on 3 separate physical machines, each hosting 4 guest virtual machines that executed the malware samples and were monitored by CAPE. These virtual machines were configured as follows:

- Windows 7 operating system was installed together with MS Office
- the virtual machines were altered to mimic a real personal computer, hence
    - A collection of typical popular applications was installed, including Google Chrome, Firefox, Adobe Reader, Spotify
    - A private key was generated and stored in a standard location
    - At least one password was stored in Chrome
    - A collection of publicly available documents and images were downloaded and stored in typical locations

- Internet connection was allowed

3.2 Data Description

The purpose of the dataset is to allow researchers to work with detailed and rich behavioral data collected from several malware families over an extended period. We first describe the metadata followed by the structure of the samples themselves.

3.2.1 Metadata

The dataset consists of 48, 976 samples. For each sample, the dataset provides:

• **sha256** of the sample for the identification,

• **classification** to a malware family,

• **type** of the malware sample,

• **date** of detection of the file.

All of these samples were classified as malicious; however, they belong to different malware families. Labeling these samples and assigning a particular malicious PE into a malware family is not an easy task as some code can be shared by various families thus causing possible noise in labels [20]. According to the records in Avast systems, the classification of these samples into 10 different malware families is provided in Table 3.1.

Adload	Emotet	HarHar	lokibot	njRAT	Qakbot	Swisyn	Trickbot	Ursnif	Zeus
704	14,429	655	4,191	3,372	4,895	12,591	4,202	1,343	2,594

Table 3. 1 Number of samples in the dataset according to malware family [21].

Moreover, the samples are one of these 6 types:

"banker", "trojan", "pws", "coinminer", "rat", "keylogger"

The samples were collected mainly throughout the years 2017 and 2019, thus offering a significant spread over time for studying the changes in the families and the changes in the data distribution.

Figure 3.1 depicts the histogram of years and months in which the samples were detected.

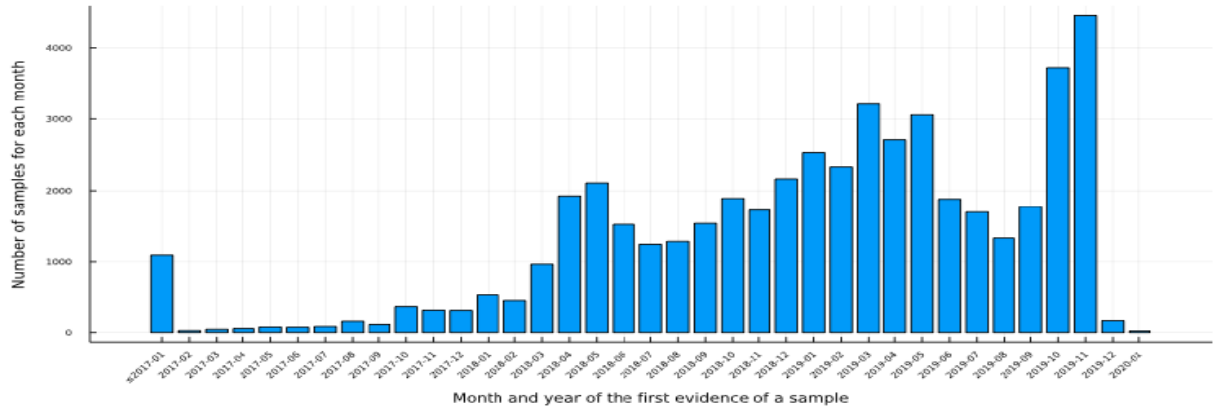

Figure 3. 1 Histogram of number of samples collected over several months in the dataset [18].

3.3 Dataset

The dataset itself consists of JSON reports produced by the CAPE sandbox. For each analyzed sample, there is a full JSON report of the CAPE sandbox, stored as a plain JSON in a file identified by the sha256 of the analyzed sample [22]. The full JSON files contain all gathered information, including all information about the started processes and system calls, all the generated (dumped) binaries, and (if any) existing detections based on YARA rulesets.

Although the key purpose for releasing the dataset is to allow the AI/ML researchers to develop and test new methods and classifiers, no transformation of JSON reports into vector representation

was provided by the authors, the main reason being that there exist methods that can take the JSON as an input without any transformation.

However, the presence of some registry keys in the full reports can leak true labels and thus invalidate using full reports directly for training ML models. Moreover, the size of the full reports can be prohibitively large – in a few cases, the size of the full report is more than 800 MB.

Figure 3.2 shows a full histogram of sizes of full reports.

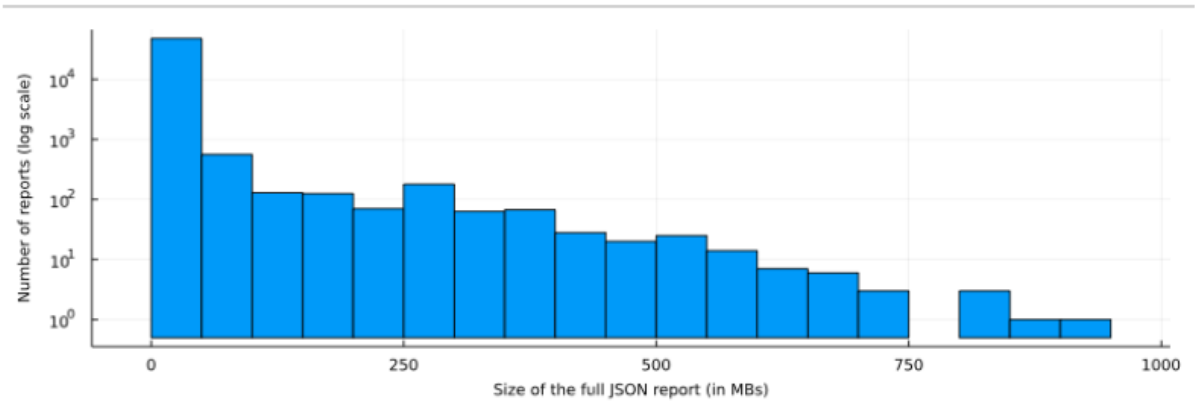

Figure 3. 2 The histogram of sizes of full JSON reports produced by CAPEv2 on samples in the dataset [19].

Therefore, besides the set of full reports, the authors have also created a set of reduced reports. They generated these reduced reports by restricting the original JSON to two keys containing enough information for training an ML classifier. These keys are:

- **behavior** "summary" that contains a summary of the behavioral analysis, including accessed
- **static** "pe" that contains static properties of the executed sample.

Figure 3.3 is the shortened example of a JSON output of CAPEv2 analysis of a malicious sample:


```
{
   "behavior": {
       "summary": {
           "keys": [
              "HKEY\_LOCAL\_MACHINE\\System\\CurrentControlSet\\Control\\Nls\\CustomLocale",
              ...],
           "resolved\_apis": [
              "kernel32.dll.GetCurrentProcessorNumber",
              "kernel32.dll.GetNativeSystemInfo",
              ...],
           "executed\_commands": [
              "\"C:\\Users\\comp\\AppData\\Local\\Temp\\FFFF450D574E5E5706FB.exe\"",
              "\"C:\\Users\\comp\\AppData\\Local\\Microsoft\\Windows\\spcmachine.exe\""],
           "write\_keys": [],
           "files": [
              "C:\\Windows\\SysWOW64\\kernel32.dll",
              "C:\\Windows\\Globalization\\Sort,
              ...],
           ...
           "mutexes": [
              "PEMB40",
              "PEM868",
              "Global\\I5C3A8244",
              "Global\\M5C3A8244",
              ...]
       }
    },
    "static": {
       "pe": {
           "icon\_hash": null,
           "sections": [
              {
                  "raw\_address": "0x00001000",
                  "name": ".text",
                  "characteristics": "IMAGE\_SCN\_CNT\_CODEIMAGE_SCN_MEM_EXECUTEIMAGE\_SCN\_MEM\_READ",
                  "virtual\_size": "0x00002786",
                  "virtual\_address": "0x00001000",
                  "size\_of\_data": "0x00003000",
                  "entropy": "5.83",
                  "characteristics\_raw": "0x60000021"
              },
           ...
           "peid\_signatures": null,
           "entrypoint": "0x00403600",
           "exports": [],
           "overlay": null,
           "digital\_signers": [],
           "imphash": "4e77bf5b96ea24734ed70b788b9fb7c8",
           "reported\_checksum": "0x00000000",
           "icon": null,
           "guest\_signers": {
              "aux\_error": true,
              "aux\_sha1": null,
              "aux\_timestamp": null,
              "aux\_valid": false,
              "aux\_signers": [],
              "aux\_error\_desc": "No signature found. SignTool Error File not valid
                 C\\Users\\comp\\AppData\\Local\\Temp\\FFFF450D574E5E5706FB.exe"
           },
           "actual\_checksum": "0x0002a738",
```


Figure 3. 3 Shortened example of a JSON output of CAPEv2 analysis of a malicious sample [23].

Chapter 4: Malware Categorization Model

4.1 Proposed Methodology

Our proposed methodology for malware categorization is centered around NLP techniques, specifically utilizing the stopword method, to prepare and vectorize the data. This preparation was crucial for applying the approach to three distinct ML models.

The process began with data preprocessing, which involved thorough feature selection. Feature selection is critical in identifying and retaining the most relevant attributes from the dataset, thereby improving the efficiency and performance of the ML models. This step ensures that the models are trained on data that is both representative and devoid of unnecessary noise.

Following the feature selection phase, the next step was data preparation. This involved cleaning the data, removing any redundant or irrelevant information, and transforming the data into a format suitable for ML algorithms. Here, the stopword technique played a key role by filtering out the common words that are usually insignificant in text analysis, allowing the models to focus on the more meaningful components of the dataset.

Once the data was properly prepared and vectorized, it was ready to be fed into the three ML models. These models were then trained to identify the patterns and characteristics associated with the malware, leveraging the refined dataset to enhance their predictive accuracy and reliability.

Figure 4.1 provides a flowchart of categorization process.

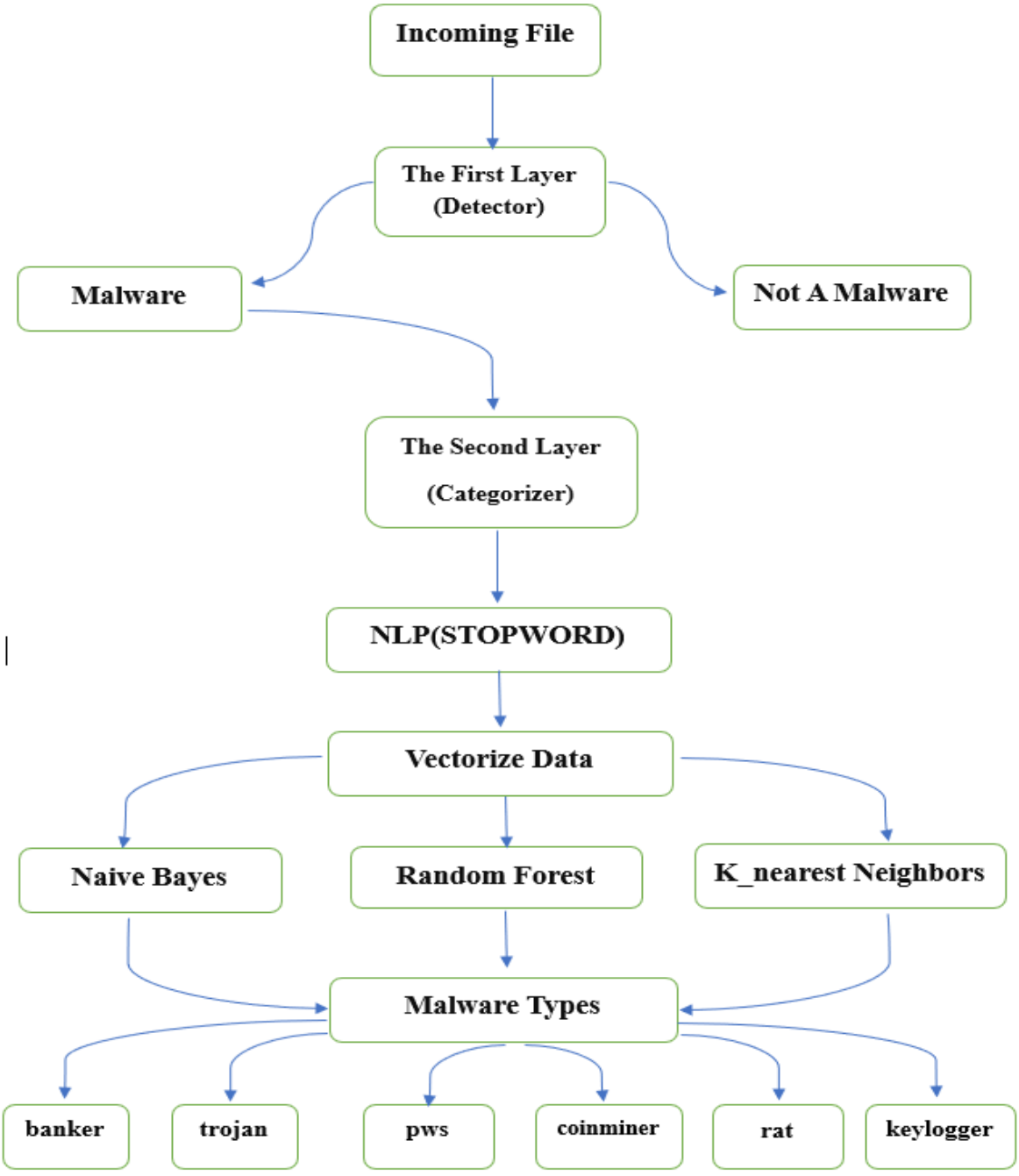

Figure 4. 1 Flowchart of the categorization process

4.2 Data Preprocessing

Since the data is in raw JSON format with a tree structure, the initial step was to thoroughly examine its organization. This involved understanding the hierarchical structure and the relationships within the data to ensure an accurate parsing and analysis.

To better comprehend the dataset, the accompanying paper was referred to, which provided valuable insights and context [24]. That paper highlighted two key features, "behavior" and "static," as particularly significant for training the model and predicting the labels. These features were identified as critical components that could significantly influence the performance and accuracy of the ML models.

Recognizing the importance of these features, the raw data was filtered. This process involved extracting and isolating the relevant sections of the JSON files that corresponded to the "behavior" and "static" features, respectively. By doing so, a new refined dataset that contained only the pertinent files was created, which was more manageable and directly aligned with the objectives of the training and label prediction tasks, hence facilitating a more efficient and targeted analysis.

To enhance the comprehension, the two datasets (i.e. the main dataset and the labels) were merged into a single, unified data frame. This integration ensured that each record in the main dataset was directly accompanied by its corresponding label, facilitating a more cohesive and accessible structure for analysis.

Merging these datasets involved carefully aligning the records based on a common identifier, ensuring that the labels match accurately with their respective data points. This step was crucial for maintaining the integrity of the data, and for ensuring that subsequent analyses would yield valid and meaningful results.

Once the datasets were merged, data cleansing procedures to prepare the data for further analysis were operated. One of the key steps in this process was the elimination of any NaN (Not a Number) values from the datasets. It should be noted that NaN values can arise from various issues such as missing data or errors during data collection, and these can significantly impact the performance of ML models. By identifying and removing these values, the dataset was ensured to be clean and reliable, reducing the potential for inaccuracies in the analysis and model training phases.

In the merged dataset, columns such as “resolved apis”, “executed_commands”, “read_files”, “started_service”, “created_service”, and others, could be removed because their data were not contributing to the training process. Indeed, these columns do not provide any meaningful information or features that the ML model can use to learn and make accurate predictions. Therefore, including them in the dataset would have add unnecessary complexity and noise.

On the other hand, some samples (i.e. rows) in the dataset lacked consistency in the columns that they contained. This inconsistency could be due to missing data, errors in data collection, or differences in the data sources. Also, these inconsistent rows were missing the two most critical columns, “private” and “statistic”, which are essential for the analysis and training process, as they likely contain key information that significantly influences the model's performance. Without these columns, the rows are incomplete and cannot be used effectively in the training process. Consequently, these rows were removed from the dataset to maintain data quality and integrity.

Additionally, some columns in the dataset contained only numeric values, all of which were zeros. These zero-valued columns do not provide any variation or useful information for the model to learn from. Since they do not contribute to distinguishing between different samples or providing any predictive power, they were considered uninformative, and removing them streamline the dataset, reducing its dimensionality and improving the efficiency of the training process. By

eliminating these redundant columns, we ensured that the dataset remains focused on the most relevant and informative features, ultimately enhancing the model's performance.

The result was a well-organized and clean dataset, ready for further exploration and analysis. This preparation laid a solid foundation for accurate and effective ML model development and other analytical tasks.

Additionally, the main dataset was merged with its associated labels using a common identifier column, 'sha256'. This merging process involved aligning the records from the main dataset with their corresponding labels based on the 'sha256' identifier, ensuring that each data entry was accurately paired with its respective label. This step was essential for creating a comprehensive dataset, where each record included the data and its associated label, resulting in a new, clean dataset ready for analysis and modeling.

In this dataset, we have 10 distinct "classification_family" categories available for prediction. These categories include 'Swisyn', 'Emotet', 'njRAT', 'Zeus', 'Qakbot', 'Ursnif', 'Trickbot', 'Lokibot', 'HarHar', and 'Adload'. Each of these families represents a specific type of malware, providing a diverse set of categories for classification tasks.

Additionally, there are 6 "classification_type" categories for prediction. These categories are 'trojan', 'banker', 'rat', 'pws', 'coinminer', and 'keylogger', and each classification type represents a broader category of malware, focusing on the general behavior or purpose of the malware rather than specific families.

Our primary focus for prediction is on the "classification_type" categories. By concentrating on these categories, we aim to develop a model capable of accurately predicting the general type of malware, which can provide valuable insights for malware detection and prevention efforts. This

focus allows us to address a broad range of malware behaviors and characteristics, enhancing the overall effectiveness and applicability of our predictive models.

4.3 Data Preparation

After data preprocessing, the next step involves preparing the input data by creating a new dataset that excludes certain columns. Figure 4.2 illustrates the removal of columns that contribute to the predicting (target columns) and the creation of the input data frame.

```
input_data = merged_df.drop(['sha256', 'classification_family', 'classification_type',
                             'date', 'timestamp'], axis=1)
input_data
```

	keys	resolved_apis	executed_commands	
0	['HKEY_LOCAL_MACHINE\\SOFTWARE\\Microsoft\\OLE...	['kernel32.dll.IsProcessorFeaturePresent', 'ke...	['c:\\windows\\system\\explorer.exe']	['HKE'
1	['HKEY_CURRENT_USER', 'DisableUserModeCallback...	['kernel32.dll.FlsAlloc', 'kernel32.dll.FlsGet...	['--81893142', 'C:\\Users\\comp\\AppData\\Loca...	
2	['HKEY_LOCAL_MACHINE\\SYSTEM\\CurrentControlSe...	['kernel32.dll.GetNativeSystemInfo', 'kernel32...	['"C:\\Users\\comp\\AppData\\Local\\Temp\\0034...	
3	['HKEY_LOCAL_MACHINE\\Software\\Microsoft\\Win...	['lpk.dll.LpkEditControl', 'kernel32.dll.FlsAl...	['"C:\\Users\\comp\\AppData\\Local\\Temp\\h21v...	['HKEY_CL

Figure 4. 2 Creating the input data by dropping the target columns

1. In the initial step, specific columns were reserved in the dataset for predicting the labels. These columns were later used for training and evaluation of the performance of the ML model. Next, we focused on preparing the data for the NLP tasks. This involved several essential steps, namely:

- Stopword Identification: Common stopwords were identified and removed from the text. Stopwords do not carry significant meaning and can be safely excluded.
- Lemmatization: To reduce the words to their root forms, lemmatization was applied. For example, the word ‘better’ would be transformed to its root form ‘Lemme’ or ‘good.’
- Vectorization: To make the data compatible for ML models, the CountVectorizer from the scikit-learn library was utilized. This technique converted the text data into a numerical representation, creating a matrix of word frequencies.

2. After text preprocessing, the labels were encoded. Each category (e.g., ‘trojan,’ ‘banker,’ ‘rat’) was assigned a numerical value. For instance, ‘trojan’ might be encoded as 1, ‘banker’ as 2, and ‘rat’ as 3.
3. Finally, the dataset was split into training and testing subsets. The last 20% of records were allocated for testing purpose, while the remaining records were allocated for model training purpose.

For a better understanding of the target data, Figure 4.3 illustrates the distribution of the labels:

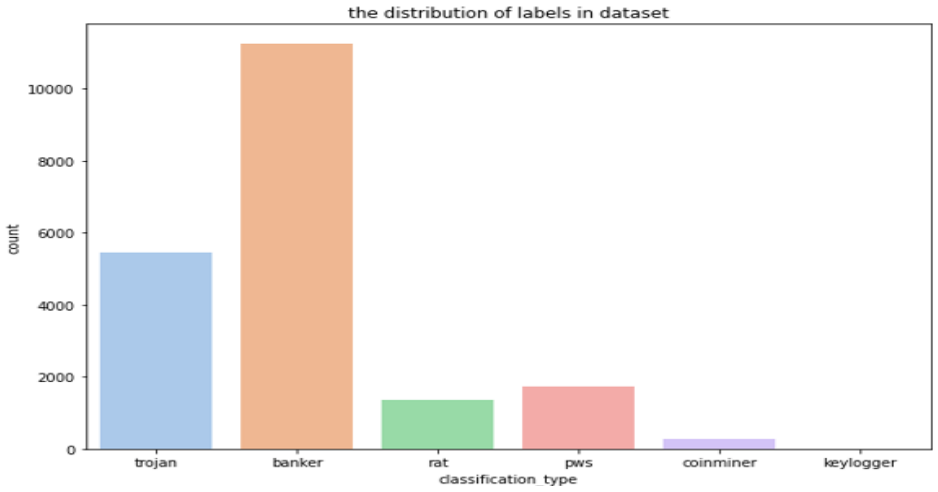

Figure 4. 3 Distribution of labels.

To ensure that the test dataset has examples from all target types (classes), the stratified sampling technique was used as follows:

1. **Calculate class distribution:** First, calculate the distribution of the target labels (classes) in the dataset. Count the number of examples belonging to each class.
2. **Determine the split ratio:** Decide on the ratio of data to allocate to the training and testing sets. We used a split 80-20% for training and testing, respectively.
3. **Stratified sampling:** Use the class distribution calculated in Step 1 to perform the stratified sampling. This means that when splitting the data into training and testing sets, ensure that each set contains a proportional representation of examples from each class. For instance, if we have three classes and we are splitting the data into training and testing sets, we make sure that both sets contain examples from all three classes in approximately the same proportions. After ensuring that each set has examples from all classes, the data is shuffled to introduce randomness. This helps prevent any bias that might occur if the data is ordered in a specific way.

By using the stratified sampling technique, we can ensure that our test dataset has examples from all target types, which is crucial for accurately evaluating the performance of the classifier.

4.4 Three Classifiers and Performance Comparison

We explore in the proposed approach three different classifiers, namely, naive Bayes, k-nearest neighbors, and random forests. Using the CAPE dataset, we train and test each of the classifiers and generate a classification report, which summarizes the standard ML metrics.

The classification report provides a detailed breakdown of the model's performance metrics, i.e. precision, recall, F1-score, and support for each class. Theses metrics are defined as follows:

- Precision: The ratio of true positive predictions to the total number of positive predictions. It indicates how many of the predicted positive instances are correct, and this is more than 0.94 in most of the classes.
- Recall: The ratio of true positive predictions to the total number of actual positive instances. It measures the model's ability to find all the relevant cases within a dataset.
- F1-Score: The harmonic mean of precision and recall. It provides a single metric that balances the trade-off between precision and recall.

4.4.1 Naive Bayes Classifier

The Naive Bayes model is a probabilistic classifier based on Bayes' theorem with the "naive" assumption that all features are independent of each other given the class label. Naive Bayes is well-suited for text classification tasks for several reasons.

This model was chosen for its computational efficiency, making it ideal for large text datasets. It is simple to understand and implement, which is great for quick prototyping and baseline comparisons. It generally performs well in text classification, particularly with a high feature-to-sample size ratio. It is also robust to irrelevant features due to its feature independence assumption. However, this assumption can lead to suboptimal performance since real-world data often feature some interdependencies. Additionally, if the test data contains features not seen in the training phase, Naive Bayes may incorrectly assign a zero probability, impacting the accuracy. Lastly, its simplicity may not capture complex feature relationships as effectively as more sophisticated

models would do, and it might struggle if the actual feature distribution within the classes differs from the model's assumptions.

The Naive Bayes model was implemented as follows:

1. The necessary libraries were first imported from scikit-learn (sklearn), a ML library in Python. Specifically, the Naive Bayes classifier was imported, which is well-suited for classification tasks involving categorical data. This classifier is based on Bayes' theorem and assumes that the features are conditionally independent given the class, making it a simple and powerful tool for many classification problems.
2. Once the libraries were imported, a Naive Bayes classifier was created. This involved initializing the classifier using sklearn's built-in functions. The classifier serves as the foundation of our model, tasked with learning from the training data to make accurate predictions.
3. Next, the training datasets were used, which include the input features and corresponding output labels, to fit the classifier. The training process involves feeding the input data into the model, allowing it to learn the relationships between the features and the target labels. This step is critical as it enables the model to understand the patterns and structures within the training data.
4. After fitting the classifier with the training data, the model was ready to make predictions. The classifier was then used to predict the labels for unseen samples. These samples were taken from the test dataset, which was kept separate from the training data to provide an unbiased evaluation of the model's performance. Predicting on the test dataset is essential as it helps assessing how well the model generalizes to new, unseen data, which is a crucial aspect of any ML model.

Figure 4.4 illustrates this entire process, starting from importing the necessary libraries and creating the Naive Bayes classifier, to fitting it with the training data, and finally using it to predict the outcomes for the test data.

```
# importing first model
from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB().fit(X_train, y_train)
```

```
# see the accuracy of naive bayes model
y_preds_naive_bayes = clf.predict(X_test)
accuracy_score(y_test, y_preds_naive_bayes)
```

```
0.935
```

Figure 4. 4 Naive Bayes Implementation.

As per Figure 4.4, the accuracy of the Naive Bayes model is approximately 93 %. This high level of accuracy indicates that the model has successfully learned the patterns and relationships within the training data and was able to generalize well to unseen data. In this context, accuracy is a measure of the proportion of correctly predicted instances among the total number of instances evaluated. Achieving an accuracy of 93% means that out of every 100 predictions made by the model, 93 of them are correct. This metric is crucial for understanding the effectiveness of the model and its potential application in real-world scenarios. Such accuracy demonstrates that the Naive Bayes model is a robust and effective tool for the given classification task, making it a valuable asset for further analysis and application.

Table 4.1 shows the classification report resulting from the model evaluation:

Models	Class	Precision	Recall	F1-Score
Naive Bayes (Accuracy 93%)	1	0.92	0.94	0.93
	2	0.98	0.95	0.96
	3	0.75	0.91	0.82
	4	0.79	0.84	0.82
	5	1.00	1.00	1.00
	6	1.00	1.00	1.00

Table 4. 1 Naive Bayes Performance Metrics

As per Table 4.1, the predicting classes 3 and 4 poses some challenges for this model. This can be seen in the lower precision, recall, and F1-scores for these classes compared to others, indicating that the model has more difficulty in correctly identifying instances of classes 3 and 4, and may be misclassifying them more frequently.

Despite these challenges, the overall accuracy of the model remains satisfactory, meanings that, on average, the model performs well across the entire dataset, correctly classifying a high proportion of instances. The high overall accuracy suggests that while there are difficulties with specific classes, the model still has a strong general performance.

4.4.2 K_nearest Neighbors Classifier

The K-nearest Neighbors Classifier was experimented as my second model choice. It is a type of instance-based learning that classifies a data point based on how its neighbors are classified. This method leverages the idea that similar data points will be found close to each other in the feature space. In implementing this model, various configurations and hyperparameters were explored to

optimize the model’s performance. Specifically, the number of neighbors k was varied to find the optimal balance between the bias and variance. Different distance metrics were also experimented, such as Euclidean and Manhattan distances, to assess their impact on the classifier's accuracy.

Cross-validation was also thoroughly conducted to ensure that the K-nearest Neighbors classifier could generalize well to unseen data. This model's simplicity and effectiveness made it a compelling choice, providing a robust baseline for comparison against more complex algorithms.

Based on the experiments conducted, the following parameters were chosen:

- n_neighbors=4,
- weights='uniform': Uniform weights. All points in each neighborhood are weighted equally.
- algorithm='auto': The algorithm used to compute the nearest neighbors. It attempts to decide the most appropriate algorithm based on the values passed to fit method.
- p=2: Power parameter for the Minkowski metric. When p=1, this is equivalent to using the Manhattan distance, and when p=2, this is equivalent to using the Euclidean distance.
- metric='minkowski': The distance metric to use for the tree. By default, it uses the Minkowski distance.

Table 4.2 shows the performance report generated for k-nearest Neighbors classifier. As shown the accuracy of this model is about 98%, which is better than the previous classifier.

Models	Class	Precision	Recall	F1-Score
K_nearest Neighbors (Accuracy 98%)	1	1.00	1.00	1.00
	2	1.00	0.99	0.99
	3	0.96	0.96	0.96
	4	0.94	0.97	0.95
	5	0.80	1.00	0.89
	6	1.00	1.00	1.00

Table 4. 2 K_nearest Neighbors Performance Metrics

It is found that, the K_nearest Neighbors model performs better across all classes, particularly excelling in classes 3 and 4, which the previous model struggled to predict accurately.

4.4.3 Random Forest Classifier

The Random Forest Classifier was experimented as third model choice. It is an ensemble learning method that operates by constructing multiple decision trees during training and outputting the mode of the classes (classification) or mean prediction (regression) of the individual trees. This approach helps improving the predictive accuracy and control over-fitting by averaging the results of various decision trees.

To try different parameters and pick the best ones for the model, techniques such as grid search or random search, combined with cross-validation, can be utilized to ensure that the selected parameters generalize well to the unseen data. Here, “grid search” was used.

Table 4.3 shows the obtained performance results.

Models	Class	Precision	Recall	F1-Score
Random Forest (Accuracy 99%)	1	1.00	1.00	1.00
	2	1.00	1.00	1.00
	3	0.96	0.97	0.96
	4	0.97	0.98	0.97
	5	1.00	1.00	1.00
	6	1.00	1.00	1.00

Table 4. 3 Random Forest Performance Metrics

As per Table 4.3, the Random Forest classifier has proven to be exceptionally effective in predicting our test data, with an accuracy rate of **99%**. This high level of accuracy indicates that the model is almost perfect in classifying the test instances correctly.

On the other hand, the K-nearest Neighbors classifier has also shown commendable performance, demonstrating a good precision, which means that when it predicts a label, it is correct a high percentage of time. As per Table 4.3, the result of the recall metric is also noteworthy, indicating that the Random Forest Classifier can identify a high proportion of actual positive instances correctly. Furthermore, the F1-score in this case, which is the harmonic mean of precision and recall, also suggests that the Random Forest classifier maintains a balance between these two metrics, making it a reliable model for our purposes. Both classifiers (i.e. K_nearest Neighbors and Random Forest) have their unique strengths, and their performance metrics suggest that they could be valuable assets in our predictive analysis toolkit. The Random Forest classifier, with its near-perfect accuracy, is particularly suited for scenarios where the cost of misclassification is high. Meanwhile, the K-nearest Neighbors classifier, with its strong precision, recall, and F1-score,

is ideal for situations where a balance between false positives and false negatives is crucial. Together, they are robust approaches to tackling complex classification problems.

Chapter 5: Conclusion

The significance of malware detection cannot be overstated in the context of modern digital security. It serves as a fundamental safeguard for computer systems, ensuring that these systems remain secure and compliant with various regulatory standards. The research presented in this project emphasized the importance of the second defensive layer, the triage phase, which is responsible for the categorization of malware types following their initial detection. This categorization is critical because it enables an appropriate response to be tailored to the specific nature of the threat. ML techniques are at the forefront of this categorization process. By utilizing existing datasets, these techniques have shown a strong potential for accurately identifying and categorizing different malware variants. This capability is crucial for enhancing the effectiveness of detection systems and enabling a swift response to new and evolving cyber threats, thereby strengthening the overall strategy for cyber defense.

Despite these advancements, the network security field faces several challenges that must be addressed through continued research. One such challenge is dataset bias, which can skew the model's performance and reduce its generalizability. Another challenge is the need for ML model's interpretability since understanding the decision-making process of such model is essential for trust and reliability.

For future works, there are several areas where improvements can be made, namely:

- Incorporating more sophisticated algorithms, such as deep learning and ensemble methods, can potentially increase the accuracy and reliability of the malware categorization.

- Enhancing the methods for extracting and selecting the most relevant features from the data can lead to more nuanced and effective models.
- The partnership between cybersecurity experts and ML researchers is crucial for enhancing the malware classification methods. Their joint efforts aim to improve the robustness and accuracy of these methods, which is essential for protecting against future cyber threats and securing digital spaces. This collaboration is fundamental to the progress and safety of the digital world.

References

[1] Muthu Dayalan, Young, C. S. (2022). Complexity and cybercomplexity. In *Cybercomplexity: A Macroscopic View of Cybersecurity Risk* (pp. 79-87). Cham: Springer International Publishing.

[2] Jovanovic, B. (2019). A not-so-common cold: Malware statistics, in: *Proceedings of the 2nd International Conference on Innovation in Engineering Technology (ICIET),* December 2019.

[3] Bailey, M., Oberheide, J., Andersen, J., Mao, Z. M., Jahanian, F., & Nazario, J. (2007). Automated classification and analysis of Internet malware. In: Proceedings of the 10th International Symposium on Recent Advances in Intrusion Detection (RAID).

[4] Landage, J., & Wankhade, M. P. (2013). Malware and malware detection techniques: A survey, in: *International Journal of Engineering Research and Technology (IJERT),* 2(12), 2278-0181.

[5] Gorment, N., Selamat, A., Lim, K., & Krejcar, O. (2023). Machine learning algorithm for malware detection, in: *Taxonomy, current challenges and future directions.* IEEE Access, PP*, 1-1.*

[6] Kang, M. G., Yin, H., Hanna, S., McCamant, S., & Song, D. (2009). Emulating emulation-resistant malware. In: *Proceedings of the 1st ACM Workshop on Virtual Machine Security* (pp. 11-22).

[7] Mangalingam, P., Samy, G. N., & Kheirudin, W. M. (2018, August 31). Systematic literature review for malware visualization techniques, in: *IEEE Symposium on Security and Privacy.*

[8] C. Willems, T. Holz, and F. Freiling. Toward Automated Dynamic Malware Analysis Using CWSandbox.IEEE Security and Privacy, 5(2), 2007.

[9] A. V. Aho and M. J. Corasick. E_cient string matching: An aid to bibliographic

search. Communications of the ACM, 18(6), June 1975.

[10] Mihai Christodorescu and Somesh Jha. Static analysis of executables to detect malicious patterns. In: *Proceedings of the 12th USENIX Security Symposium, Washington,D.C.*, USENIX Association, August 2003.

[11] Rodehorst, V., & Koschan, A. (2006). Comparison and evaluation of feature point detectors. In: *Proceedings of the 5th International Symposium Turkish-German Joint Geodetic Days.*

[12] Denning, P. J. (1990). Computers under attack: Intruders, worms and viruses. Reading, MA: Addison-Wesley.

[13] Bacchelli, A., D'Ambros, M., & Lanza, M. (2010). Extracting source code from e-mails. In: *Proceedings of the 18th International Conference on Program Comprehension (ICPC 2010)* (pp. 24–33).

[14] Sanchez, G. (2014, April-May). Getting data from the web, in: *Part 5, Handling JSON data*. licensed under CC BY-NC-SA 4.0.

[15] Raj, S., & Walia, N. K. (2020). A study on Metasploit Framework: A pen-testing tool. In: *2020 International Conference on Computational Performance Evaluation (ComPE).* IEEE.

[16] Dominik Kouba. *Analyzing the execution of malware in a sandbox using hierarchical multiple instance learning*. Master's Thesis, Czech Technical University in Prague, Faculty of Electrical Engineering. 2020.

[17] Moser, A., Kruegel, C., and Kirda, E. 2007, Limits of static analysis for malware detection, in: *23rd Annual Computer Security Applications Conference (ACSAC)*

[18] Bosansky, B., Kouba, D., Manhal, O., & Sick, T. (2022). Avast-CTU Public CAPE Dataset, in: *Artificial Intelligence Center, Dept. of Computer Science,*

Faculty of Electrical Engineering, Czech Tecnical University in Prague

[19] Manuel Egele, Theodoor Scholte, Engin Kirda, Christopher Kruegel, Survey Center: A Survey on Automated Dynamic Malware Analysis Techniques and Tools, in: *ACM Computing Surveys Journal, February 2012*

[20] Y. Ki, E. Kim, and H. K. Kim. A novel approach to detect malware based on api call sequence analysis. *International Journal of Distributed Sensor Networks*, *11(6):659101, 2015.*

[21] D.-G. Marculet, R. Benchea, and D. T. Gavrilut. Methods for training neural networks with zero false positives for malware detection. In *2019 21st International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC),* pages 230–236, 2019.

[22] S. K. Shaukat and V. J. Ribeiro. Ransomwall: A layered defense system against cryptographic ransomware attacks using machine learning, in: *2018 10th International Conference on Communication Systems & Networks (COMSNETS),* pages 356–363, 2018.

[23] J. Singh and J. Singh. A survey on machine learning-based malware detection in executable files. *Journal of Systems Architecture,* 112:101861, 2021.

[24] Q. Le, O. Boydell, B. Mac Namee, and M. Scanlon. Deep learning at the shallow end: Malware classification for non-domain experts. *Digital Investigation*, 26: S118–S126, 2018.