Development and Evaluation of a Machine Learning-Based Host Intrusion Detection System Using Malware and Benign Program Characteristic

Robert Geller
Alin Sălăgean
Mihai Cîra
UVT, Data Science in Industry

January 2024

Abstract

This paper discusses the development of a Host Intrusion Detection System (HIDS) utilizing machine learning to distinguish between malware and non-malicious executable programs. We created datasets from two groups of files: one group consisting of malware and the other of benign files. Key features like Machine, SizeOfOptionalHeader, and Characteristics were extracted and analyzed. The model was trained and validated using a merged set of malware and benign files, split 80%-20% for training and validation, respectively. A separate merged set was used for testing. The system's adaptability is further demonstrated by its capability to assess individual files through their feature arrays, showcasing the effectiveness of machine learning in identifying cybersecurity threats.

Contents

1 Introduction

Our project presents a detailed exploration of developing a Host Intrusion Detection System (HIDS) using two main technological tools: a sophisticated feature extractor and a K-Nearest Neighbors (KNN) machine learning model. The feature extractor plays a pivotal role, meticulously analyzing each file in the dataset to gather a comprehensive set of features. These include various header sizes, machine characteristics, and other essential attributes that uniquely identify each file. The data extracted forms the basis for the KNN model, a chosen algorithm for its efficiency in classification tasks. This model is adept at distinguishing between benign and malicious files, trained on the nuanced differences in the feature sets. The integration of the feature extractor with the KNN model is a strategic approach to accurately classify and predict the nature of files, thereby enhancing the detection capabilities of our Host Intrusion Detection System.

2 Application workflow

This section outlines the systematic flow of our application, showcasing how each component interconnects to form a comprehensive Host Intrusion Detection System. The process begins with dataset collection and moves through various stages including feature extraction, model training, and application of the model on datasets and individual files. This workflow is essential in understanding the operational structure and the sequential steps that lead to the effective detection of malware. The sequence diagram below (1), also reflects the workflow of the application.

- **Collecting the Dataset.** The initial stage involves gathering a diverse set of files. This includes collecting benign files from personal computers and obtaining malicious files from academic sources. The selection is aimed at covering a wide spectrum of file types and behaviors for a robust dataset.
- **Using the Feature Extractor to Generate Dataset.** Once the files are collected, the feature extractor is employed. This tool meticulously analyzes each file, extracting key features like file sizes, headers, and technical characteristics. The output is a detailed dataset, ready for the next stage of model training.
- **Training the Model.** With the dataset prepared, we proceed to train the K-Nearest Neighbors (KNN) model. This phase involves feeding the extracted features into the model, allowing it to learn and identify patterns that differentiate benign files from malware. The training process is crucial for the model's accuracy and reliability.
- **Using the Model on a Dataset.** Post-training, the model is then applied to a separate dataset. This dataset, which the model hasn't encountered during training, is used to evaluate the model's effectiveness in real-world scenarios, assessing its ability to correctly classify files as either benign or malicious.
- **Using the Model on a Single File.** The final step in the application workflow is the model's capability to analyze individual files. Here, a user can input a single file, and the model applies its learned knowledge to determine whether the file is benign or malicious. This functionality is critical for real-time analysis and on-the-fly security assessments

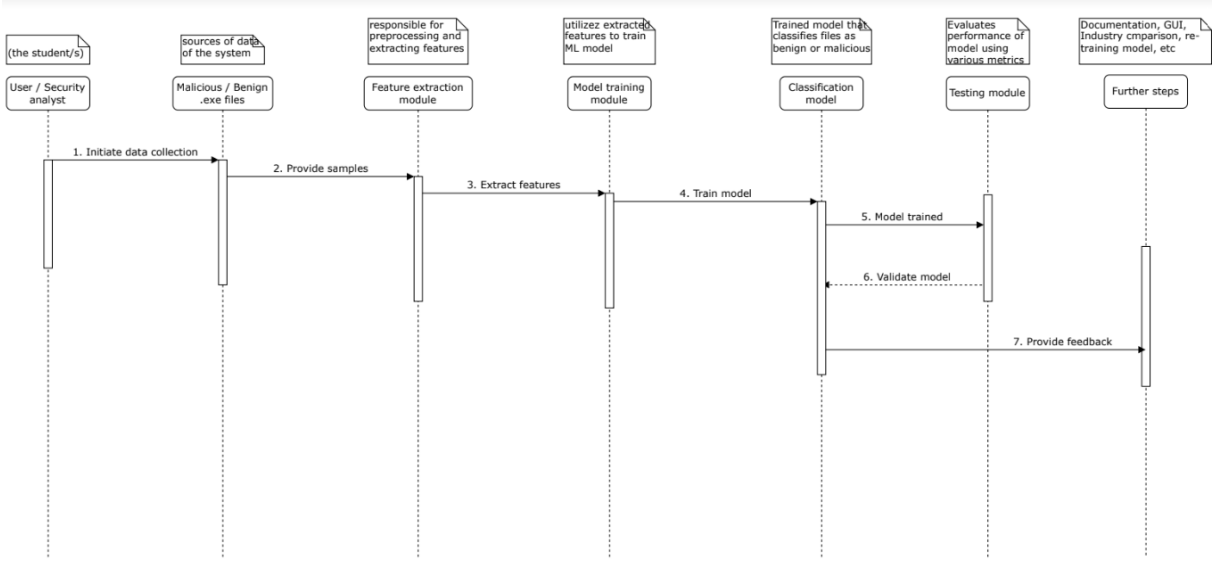

Figure 1: Sequence diagram of the workflow

3 The dataset

Before delving into the specifics of the benign and malicious files, it is pertinent to note that the combined dataset used for analysis in this project comprises approximately 3400 entries. These entries represent a diverse collection of files, providing a comprehensive basis for our machine learning model to learn and distinguish between different types of file behaviors. In the figure below (2) we can see the distribution of the dataset based on the class feature.

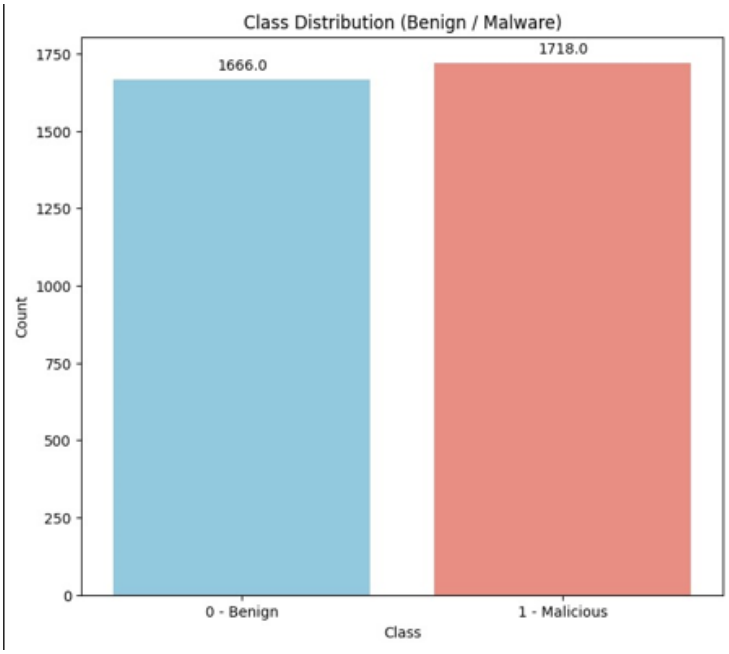

Figure 2: Class distribution of the dataset

3.1 Benign Files

The benign portion of our dataset originates from personal computing devices. It encompasses a diverse range of operating system executables and other non-malicious files. These files were specifically chosen to represent a normal computing environment, ensuring that the KNN model is exposed to typical user data. This approach is fundamental for the model to learn and differentiate normal file behaviors and characteristics from those indicative of malware.

3.2 Malicious Files

For the malicious files, we sourced our dataset from an academic perspective, provided by the course instructor. This dataset includes various types of malware, offering a comprehensive and challenging array of data for our HIDS. The inclusion of these files is crucial for testing the detection and classification prowess of our KNN model. This subsection details the characteristics of these malicious files and the importance of their diversity in training a robust machine learning-based intrusion detection system.

4 Feature Extraction

The feature extraction process is a critical component of our Host Intrusion Detection System. For each file in our dataset, a specific Python script is executed to extract a set of features. These features are essential for the subsequent machine learning model to accurately classify files. The extracted data from each file is then compiled into a CSV file using another Python script, forming the dataset for our project.

The following Python code snippet highlights the main components of the feature extraction process:

Listing 1: Python Code for Feature Extraction

```
# Python code snippet for feature extraction
import ...

def extract_features(file_path):
    # Load the file for analysis
    ...
    # Extract various features
    Machine = ...
    SizeOfOptionalHeader = ...
    Characteristics = ...
    ...
    # More feature extraction
    ...

    return feature_vector

# Loop over each file for feature extraction
for file in dataset:
    features = extract_features(file)
    # Code to append features to a CSV file
    ...
```

Key components of this code include:

- **Feature Extraction Function:** The function `extract_features` is designed to process an individual file and extract various features such as Machine, SizeOfOptionalHeader, Characteristics, etc.
- **Loading and Analyzing Files:** The script loads each file and performs a detailed analysis to extract the required features.

- **Creating the Feature Vector:** Each file's features are compiled into a feature vector, which is then used for machine learning model training.
- **Compiling into CSV:** The extracted features from all files are accumulated and appended to a CSV file, effectively constructing the dataset for the model.

This extraction process is repeated for each file in the dataset, ensuring a thorough and comprehensive analysis of all files, both benign and malicious.

4.1 Feature example: The Entropy

Among the numerous features extracted from each file, entropy stands out as a particularly important metric. In the context of file analysis, especially for intrusion detection, entropy is a measure of randomness or disorder within the file's contents. High entropy is often indicative of encrypted or compressed data, which is a common characteristic in many types of malware. Conversely, lower entropy is typical of more structured or predictable data.

The entropy calculation is a crucial part of the feature extraction process, as it provides insights into the nature of the file being analyzed. High entropy values might signal an attempt to disguise malicious activities, while lower values could suggest regular, benign files. This feature, therefore, plays a vital role in our machine learning model's ability to differentiate between malicious and non-malicious files.

Here is a simplified version of the Python code used to calculate the entropy of a file:

Listing 2: Python Code for Entropy Calculation

```
import math

def calculate_entropy(data):
    # Frequency of each byte in the file
    frequency = [0]*256
    for byte in data:
        frequency[byte] += 1

    # Entropy calculation
    entropy = 0
    for freq in frequency:
        if freq > 0:
            probability = freq / len(data)
            entropy += -probability * math.log2(probability)
    return entropy
```

In this code:

- The function `calculate_entropy` computes the entropy of a given file.
- It first calculates the frequency of each byte value (0-255) in the file.
- The entropy is then calculated using the formula, which involves the probability of each byte and the logarithm function, reflecting the level of disorder or randomness in the file's data.

By analyzing the entropy of files, our system gains an additional layer of analysis, enhancing its capability to identify potential threats and anomalies in the data.

5 The Models

In our Host Intrusion Detection System, the primary machine learning model utilized is K-Nearest Neighbors (KNN). However, to enhance performance and for research purposes, we also integrated the HistGradientBoosting classifier. This section briefly describes the implementation and workflow of these two models, highlighting their distinctive roles and functionalities in our system.

5.1 K-Nearest Neighbors (KNN)

The KNN model in our project follows a series of steps, starting with loading the dataset and performing preprocessing. Preprocessing includes handling missing values and normalizing the features for more efficient learning. Specifically, missing values in the dataset are filled with the mean of their respective class, a technique that helps in maintaining the integrity of the data. After imputation, the features are normalized using standard scaling to ensure they are on a comparable scale, which is crucial for distance calculation in KNN.

The dataset is then split into training and testing sets, following an 80-20 ratio. The KNN classifier is built and trained on the training set. Model accuracy is evaluated on the test set to assess its performance in classifying files as benign or malicious. Additionally, the model is equipped to make predictions on new, unseen data, applying the same preprocessing steps before making a prediction.

5.2 HistGradientBoosting

For the HistGradientBoosting classifier, the workflow begins similarly by loading the dataset and splitting it into features and target variables. The dataset is divided into training and testing sets to validate the model's performance after training.

The HistGradientBoosting classifier is then instantiated and trained on the training data. This model is particularly effective for large datasets and complex feature interactions, making it a valuable addition to our project. Post-training, the model's accuracy is computed on the test set, providing insights into its effectiveness in distinguishing between benign and malicious files.

Like the KNN model, HistGradientBoosting also supports making predictions on individual inputs. The model takes in new data, processes it according to the trained model's requirements, and outputs a prediction, indicating the presence or absence of malware.

Both models play a crucial role in our system, each contributing to a more robust and reliable malware detection process in the Host Intrusion Detection System.

5.3 Results of Training

In the "Results of Training" subsection, we present and analyze the outcomes of training our models, focusing on the performance as indicated by the confusion matrices for both KNN and HistGradientBoosting classifiers.

For the KNN model, the confusion matrix revealed a significant number of false negatives and false positives. This indicates that, while the model was able to correctly identify several instances, it also misclassified a notable amount of malicious files as benign (false negatives) and benign files as malicious (false positives). This outcome suggests limitations in the model's ability to differentiate between the two classes under certain conditions.

In contrast, the HistGradientBoosting model exhibited a markedly improved performance. The confusion matrix for this model showed minimal false negatives and false positives. This superior accuracy in correctly classifying both malicious and benign files underscores the effectiveness of HistGradientBoosting in dealing with the complexities inherent in malware detection. The reduced error rates reflect the model's robustness and its capability to capture nuanced patterns in the data more effectively than the KNN model.

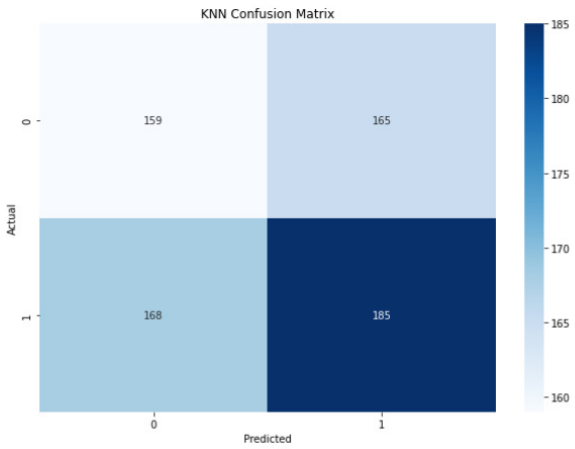

Figure 3: Confusion matrix of KNN

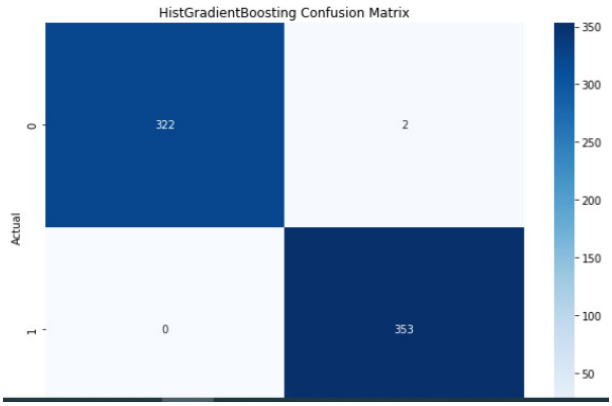

Figure 4: Confusion matrix of Histogram Gradient Boosting

These results highlight the strengths of using advanced ensemble methods like HistGradientBoosting in complex classification tasks and underscore the importance of choosing the right model for specific types of data and objectives.

6 Predictions

The "Predictions" section of our project demonstrates the application of the K-Nearest Neighbors (KNN) and HistGradientBoosting models in real-world scenarios. This phase is crucial as it reflects the practical effectiveness of our models in identifying whether a single file is malware or benign.

The prediction process involves several key steps:

Feature Extraction for Individual Files: Initially, features are extracted from individual executable files. This is done using a feature extraction method tailored to analyze and retrieve a set of characteristics from the file, mirroring those used in the training phase of the models.

Model Initialization and Training: The trained models (KNN and HistGradientBoosting) are initialized using the data from a comprehensive dataset previously prepared. This step is crucial as it equips the models with the necessary knowledge and patterns learned from the dataset, enabling them to make informed predictions.

Real-Time Predictions: With the models ready, the system is set to analyze new, individual files. The models receive the extracted features from these files and predict their nature. The prediction process involves assessing whether the characteristics of the file align more closely with those of known malware or benign files, as learned during the training phase.

Interpreting Prediction Outcomes: The outcomes of these predictions indicate the models' assessments. A file flagged as malware suggests that its characteristics strongly resemble those of malicious software, while a benign classification implies the file is likely safe and does not display typical malware attributes.

This predictive capability is a cornerstone of our Host Intrusion Detection System, offering a proactive tool in cybersecurity efforts. It showcases the models' ability to not only learn from historical data but also apply this knowledge effectively to new and unseen files, enhancing the system's reliability and responsiveness in identifying potential cyber threats.

7 Conclusions

In this project, we compared the K-Nearest Neighbors (KNN) and HistGradientBoosting models for a Host Intrusion Detection System. The comprehensive analysis, beyond confusion matrices, revealed distinct performance characteristics of each model.

KNN, while useful in certain scenarios, exhibited limitations in malware detection, likely due to its simpler, distance-based algorithm which may not effectively handle complex data patterns.

HistGradientBoosting, however, demonstrated superior accuracy with significantly fewer misclassifications. Its advanced capabilities in handling complex, non-linear relationships and its efficiency with large datasets contribute to its robust performance. The model's iterative learning approach further ensures continual adaptation and refinement.

The evident effectiveness of HistGradientBoosting in our system highlights the importance of choosing the right model in machine learning, especially for critical applications like cybersecurity.

8 References

[1] Guo, G., Wang, H., Bell, D., Bi, Y., & Greer, K. (2003). *KNN Model-Based Approach in Classification.* In *Lecture Notes in Computer Science* (pp. 986–996). doi:10.1007/978-3-540-39964-3_62.

[2] Van der Aalst, W. M. P., Batagelj, V., Ignatov, D. I., Khachay, M., Kuskova, V., Kutuzov, A., ... Tutubalina, E. (Eds.). (2019). *Analysis of Images, Social Networks and Texts.* In *Lecture Notes in Computer Science*. doi:10.1007/978-3-030-37334-4.