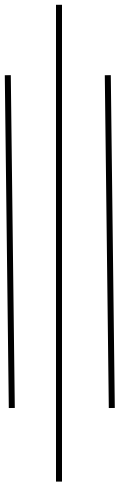

FUSEMACHINE

A PROJECT REPORT ON

“Malware Memory Analysis for Intrusion Detection”

SUBMITTED BY:

Samman Shrestha

JESTHA, 2080

Abstract

The project titled "Malware Memory Analysis for Intrusion Detection" aims to develop a machine learning-based approach for classifying memory dumps as benign, spyware, trojan horse, or ransomware. The significance of this project lies in the detection and prevention of potential cyber-attacks by employing malware memory analysis, which provides a means to identify and mitigate threats.

The dataset used in this project was obtained from the database of the University of New Brunswick and consists of a balanced collection of obfuscated malware samples, including spyware, trojan horse, and ransomware, along with benign memory dumps. Various machine learning algorithms are applied to train and evaluate classification models. The logistic regression model is initially employed, providing an accuracy score of 74%. Subsequently, a decision tree model is trained, revealing overfitting issues due to its high depth. To address overfitting, cost complexity pruning is applied, resulting in a reduced depth of 32 and an improved testing accuracy giving 85%. The support vector machine model is then employed, yielding an accuracy of 74%. The random forest classifier, an ensemble method for bagging, demonstrates superior performance with a testing accuracy of 87%. Finally, XGBoost, a boosting algorithm, achieves an accuracy of 86% on the testing data.

Contents

Lists Of Tables

Lists Of Figures

Introduction

1. Background

The proliferation of malware poses a significant threat to cybersecurity. Malware, short for malicious software, is designed to compromise computer systems, steal sensitive information, and disrupt normal operations. Traditional methods of malware detection and prevention often struggle to keep pace with the evolving sophistication of malware techniques. Malware memory analysis involves the examination of the memory contents of a compromised system to detect the presence of malware. By analyzing the memory dumps, which are snapshots of the system's memory at a given point in time, security analysts can uncover hidden malware, even when it employs obfuscation techniques to evade traditional detection methods. To address the challenges posed by obfuscated malware, the dataset incorporates memory dumps collected in debug mode. This mode ensures that the memory dump process remains concealed, simulating a real-world scenario where an average user may be unaware of the presence of malware. By capturing the system state during a malware attack, the dataset reflects a more accurate representation of the memory contents and facilitates the development of effective malware detection systems.

2. Problem Statement

- The increasing sophistication of malicious software (malware) poses a challenge for traditional detection methods and security measures, as malware employs advanced techniques to evade detection and remain undetected.
- Malware, especially when obfuscated, can compromise the integrity of computer systems, facilitate unauthorized access to sensitive information, and result in financial losses, reputation damage, and legal implications.

3. Objective

The main objective of this project is:

- Develop a machine learning model that can accurately classify memory dumps as benign or belonging to specific malware categories (spyware, trojan horse, or ransomware), thereby improving intrusion detection capabilities and enhancing cybersecurity measures.

Literature Review

Malware memory analysis and intrusion detection have been extensively studied in the field of cybersecurity. Researchers have employed various techniques and methodologies to detect and prevent cyber threats.

A survey conducted by Idika of malware detection techniques provides an overview of different malware detection techniques, including signature-based detection, behavior-based detection, and anomaly detection. It discusses the advantages and limitations of each technique and highlights the importance of memory analysis in detecting obfuscated malware.[1]

Research on Detection of obfuscated mobile malware with machine learning and deep learning models investigates the application of machine learning techniques, such as ensemble methods and support vector machine, for malware detection. The authors demonstrate the capability of machine learning models to learn complex patterns and identify malware samples based on memory analysis.[2]

Methodology

1. Data Analysis

The dataset obtained from the University of New Brunswick's database for Malware Memory Analysis was first analyzed. The dataset consists of memory dumps collected from various malware samples, including spyware, ransomware, and trojan horse malware. It is a balanced dataset with an equal distribution of benign and malicious memory dumps.

In total the dataset consists of 57 columns. Some of the features present in the dataset are:

1.1 Process-related Features:

- pslist.nproc: The number of processes in the system.
- pslist.nppid: The number of parent processes for each process.
- pslist.avg_threads: The average number of threads per process.
- pslist.nprocs64bit: The number of 64-bit processes in the system.
- pslist.avg_handlers: The average number of handlers per process.

1.2 DLL-related Features:

- dlllist.ndlls: The number of DLLs (Dynamic Link Libraries) loaded in the processes.
- dlllist.avg_dlls_per_proc: The average number of DLLs loaded per process.

1.3 Handles-related Features:

- handles.nhandles: The total number of handles in the system.
- handles.avg_handles_per_proc: The average number of handles per process.
- handles.nport: The number of port handles.
- handles.nfile: The number of file handles.
- handles.nevent: The number of event handles.
- handles.ndesktop: The number of desktop handles.
- handles.nkey: The number of registry key handles.
- handles.nthread: The number of thread handles.

- handles.ndirectory: The number of directory handles.
- handles.nsemaphore: The number of semaphore handles.
- handles.ntimer: The number of timer handles.
- handles.nsection: The number of section handles.
- handles.nmutant: The number of mutant handles.

1.4 LdrModules-related Features:

- ldrmodules.not_in_load: The number of modules not in the load state.
- ldrmodules.not_in_init: The number of modules not in the init state.
- ldrmodules.not_in_mem: The number of modules not in the memory state.
- ldrmodules.not_in_load_avg: The average number of modules not in the load state per process.
- ldrmodules.not_in_init_avg: The average number of modules not in the init state per process.
- ldrmodules.not_in_mem_avg: The average number of modules not in the memory state per process.

1.5 Malfind-related Features:

- malfind.ninjections: The number of memory injections detected.
- malfind.commitCharge: The commit charge of the injected memory.
- malfind.protection: The protection level of the injected memory.
- malfind.uniqueInjections: The number of unique injections detected.

1.6 Psxview-related Features:

- psxview.not_in_pslist: The number of processes not found in the pslist.
- psxview.not_in_eprocess_pool: The number of processes not found in the eprocess pool.
- psxview.not_in_ethread_pool: The number of processes not found in the ethread pool.
- psxview.not_in_pspcid_list: The number of processes not found in the pspcid list.

- psxview.not_in_csrss_handles: The number of processes not found in the csrss handles.
- psxview.not_in_session: The number of processes not found in the session.
- psxview.not_in_deskthrd: The number of processes not found in the deskthrd.
- psxview.not_in_pslist_false_avg: The average number of processes not found in the pslist per process.

1.7 Labels:

- Class: benign or malicious
- Category: Benign, ransomware, spyware or torjan horse

Below graph shows the number of benign and malicious samples:

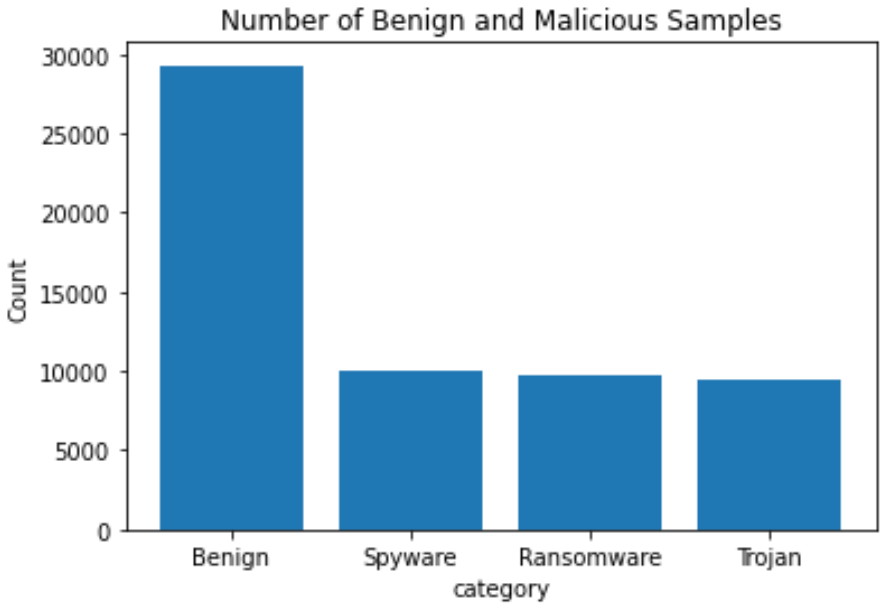

Figure 1 Graph of benign and malicious samples

2. Data Preprocessing

In the data preprocessing phase, several steps were performed to prepare the dataset for training machine learning models.

2.1 Checking for Null Values: The dataset was examined for any missing or null values. Dataset did not contain any null values.

2.2 Dropping Unwanted Columns: Next, any columns that were not relevant to analysis or classification task was identified and dropped to reduce unnecessary noise and improve computational efficiency.

2.3 Visualizing Outliers: To identify outliers in the numerical columns of our dataset, histogram plots and distribution plots was used. These plots helped to visualize the distribution of data points and detect any extreme or unusual values that could be considered outliers.

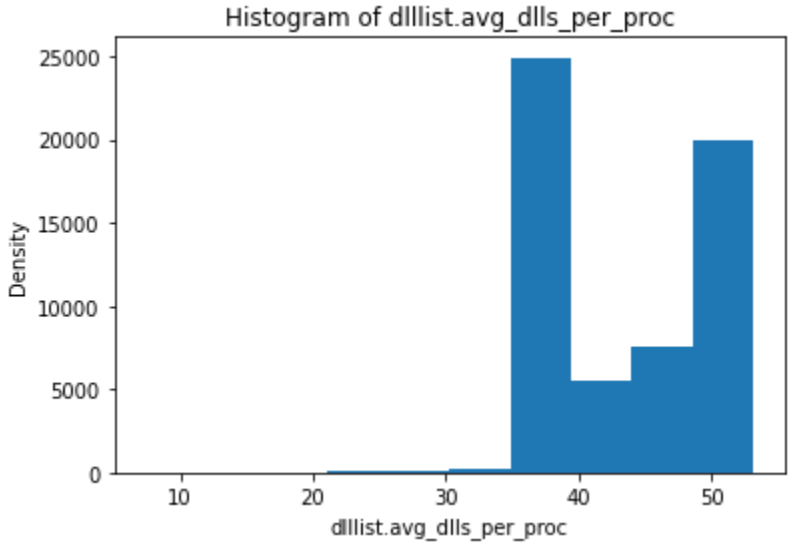

Figure 2 Histogram of dlllist.avg_dlls_per_proc

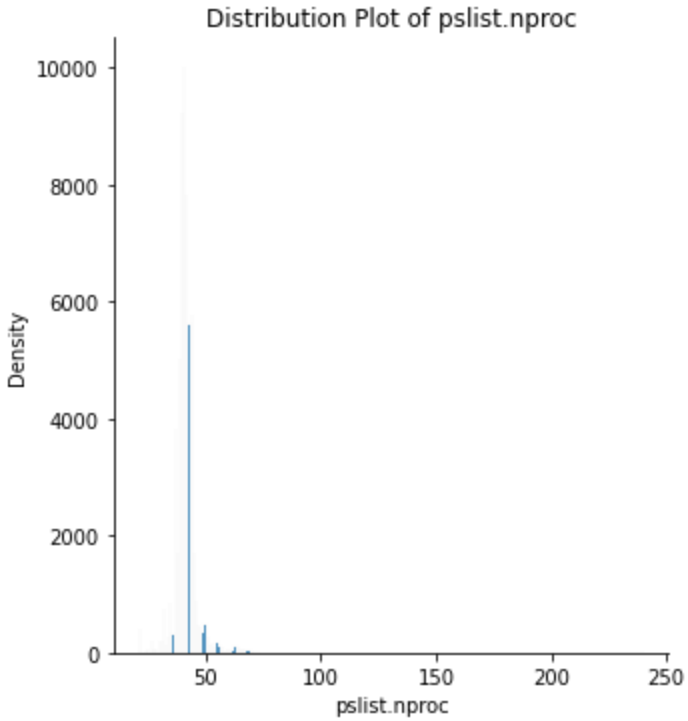

Figure 3 Distribution plot of pslist.nproc

2.4 Removing Outliers: After visualizing the outliers, Interquartile Range (IQR) method was employed to remove the outliers from the numerical columns.

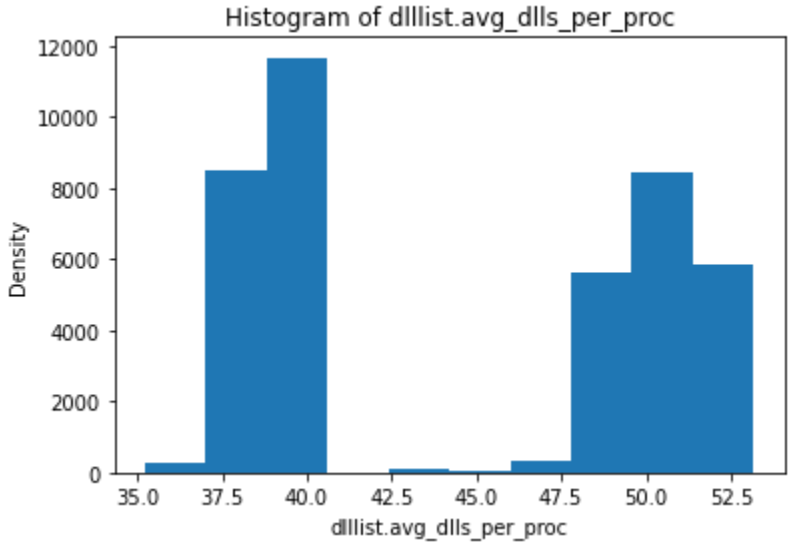

Figure 4 Histogram of dlllist.avg_dlls_per_proc after removing outlier

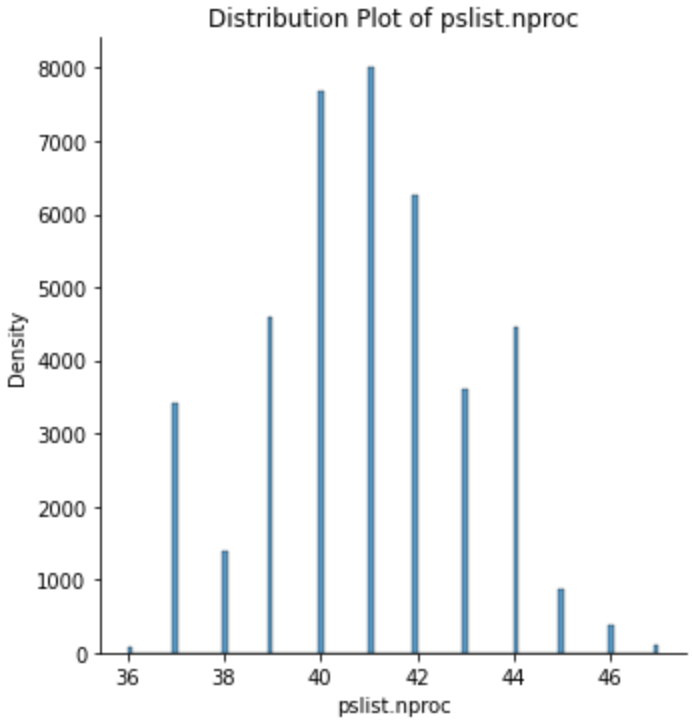

Figure 5 Distribution plot of pslist.nproc after removing outlier

2.5 Level Encoding: Dataset contained a categorical column called "category class," which represented different malware types. To convert these categorical values into numerical form suitable for our machine learning models, level encoding was performed using label encoder from sklearn. Encoded values:

Benign = 0

Spyware = 1

Ransomware = 2

TrojanHorse = 3

2.6 Train-Test Split: To evaluate the performance of our machine learning models, we split the preprocessed dataset into training and testing sets using the train-test split method from the sklearn library.

2.7 Feature Selection based on Correlation Analysis: To further refine our feature set, correlation analysis on the remaining numerical columns was performed. The correlation coefficients between each feature and the target variable (category). Features with a correlation coefficient greater than 0.75 were considered highly correlated and were retained for model training.

3. Model Training

After preprocessing the data, several machine learning models was trained on our dataset. The goal was to develop accurate models that could classify memory dumps into different malware types based on the available features. We used the following models for training:

3.1 Logistic Regression: Logistic regression uses a logistic function to model the probability of a binary outcome. A logistic regression model on the preprocessed data was trained and evaluated its performance using F1 score as evaluation metric.

3.2 Decision Tree: Decision trees use a tree-like model of decisions and their possible consequences. A decision tree model on the preprocessed dataset was trained and assessed its performance. However, the decision tree was prone to overfitting, as indicated by a high training accuracy but a relatively lower testing accuracy.

3.3 Cost Complexity Pruning: To address the overfitting issue observed in the decision tree model, we applied cost complexity pruning. This technique involves systematically varying the complexity parameter (ccp_alpha) to find the optimal trade-off between tree complexity and accuracy. A cost complexity pruning path was constructed and selected a range of ccp_alpha values. By training multiple decision tree models with different ccp_alpha values, we obtained a set of pruned trees with varying depths.

3.4 Support Vector Machine (SVM): SVM aims to find an optimal hyperplane that separates the data points of different classes. We trained an SVM model on the preprocessed data and evaluated its performance.

3.5 Random Forest: The random forest classifier, an ensemble method that combines multiple decision trees was used. Random forest creates a set of decision trees on different random subsets of the data and aggregates their predictions. We trained a random forest model on the preprocessed dataset and evaluated its accuracy on both the training and testing sets. Important feature were identified by plotting a feature importance plot and unimportant feature were dropped.

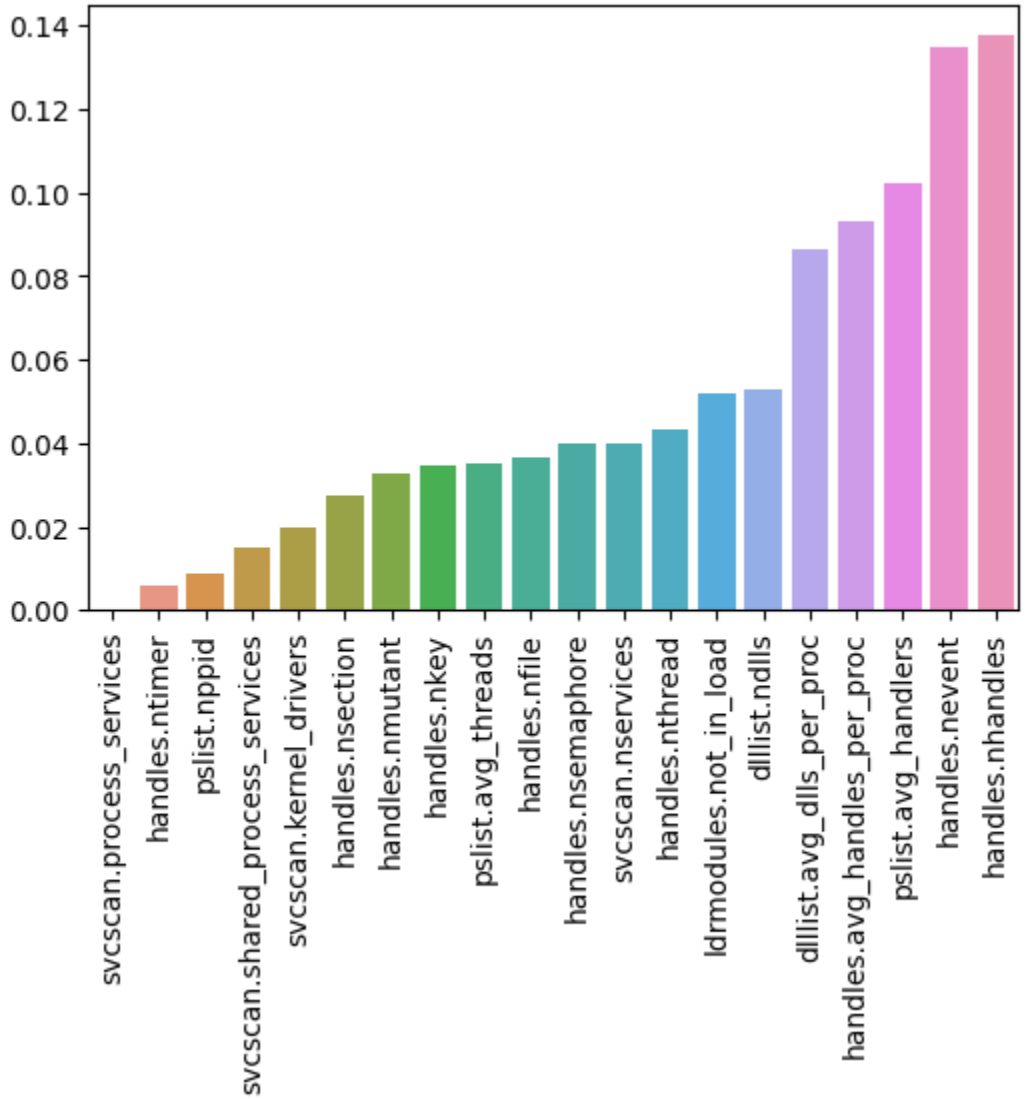

Figure 6: Feature importance plot using random forest classifier

3.6 XGBoost: XGBoost, is an optimized gradient boosting framework known for its high performance. XGBoost sequentially trains weak learners and combines their predictions to create a strong model. An XGBoost model on the preprocessed dataset was trained and evaluated its f1 score on both the training and testing sets.

By training multiple models and evaluating their performance, we aimed to identify the model that provided the highest accuracy and robustness in classifying memory dumps into different malware types.

Results and Discussion

After training and evaluating multiple machine learning models on our dataset, we obtained the following accuracy scores for each model:

Model	Training F1 score	Testing F1 score
Logistic Regression	0.74512	0.74028
Decision Tree	0.9982	0.84436
Decision Tree (Pruned)	0.96243	0.85369
Support Vector Machine	0.7508	0.74945
Random Forest	0.99825	0.8724
XGBoost	0.92156	0.86933

Table 1 F1 score of all models

Heatmap obtained from various model on testing dataset is shown below:

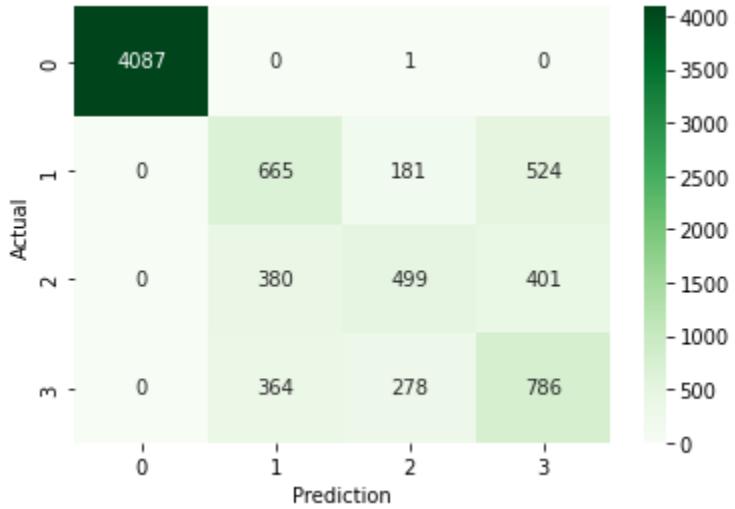

Figure 7 heatmap for logistic regression model

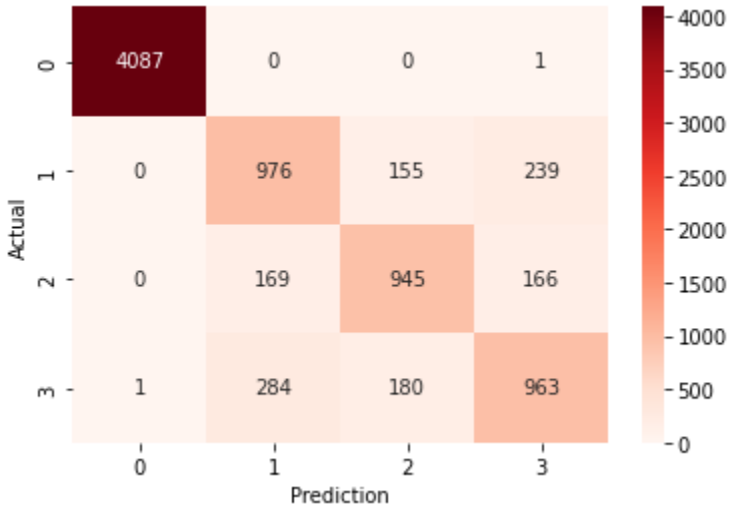

Figure 8 heatmap for decision tree model(pruned)

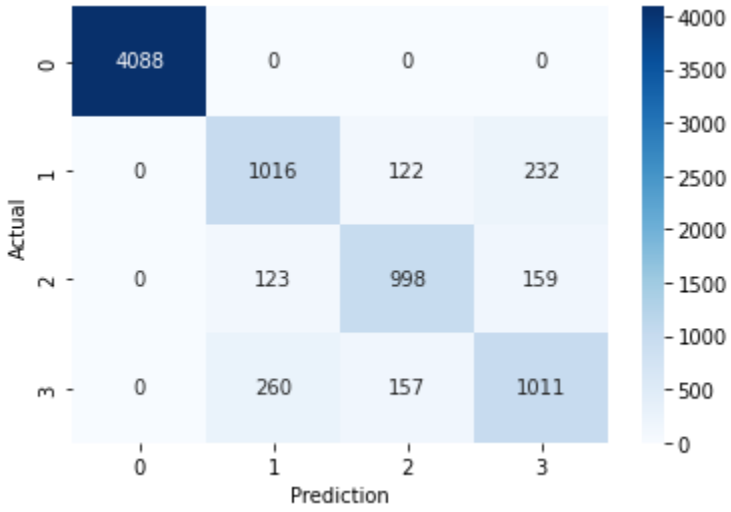

Figure 9 Heatmap for random forest classifier

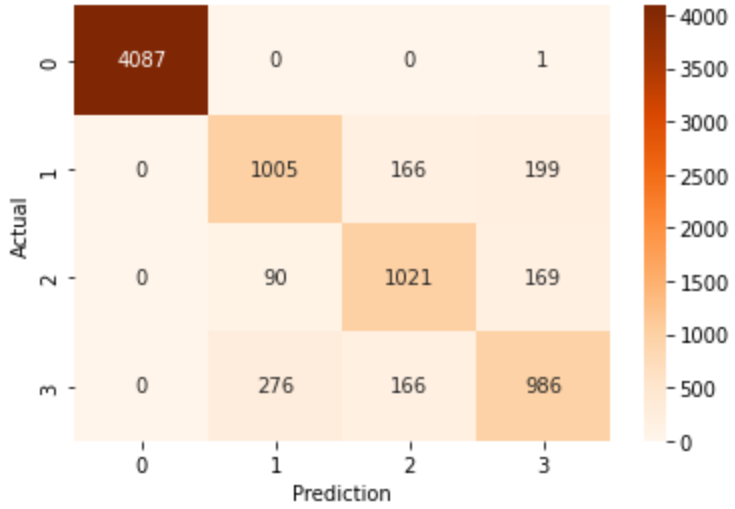

Figure 10 Heatmap for XGboost

The random forest model outperformed other models with a training f1 score of 99.83% and a testing f1 score of 87.24%. This indicates that the random forest model effectively learned the patterns in the data and generalized well to unseen instances.

Conclusion

In this project, we aimed to perform malware memory analysis for intrusion detection using machine learning techniques. Through data preprocessing, feature selection, and model training, we achieved promising results in classifying memory dumps into different malware types.

The random forest model emerged as the most accurate model, with a F1 score of 87.24%. This highlights the potential of machine learning in effectively identifying and categorizing malware based on memory data. By addressing overfitting through cost complexity pruning, we improved the generalization performance of the decision tree model.

Overall, our findings demonstrate the value of machine learning in detecting and classifying malware from memory dumps. Further research can focus on refining the models, exploring alternative feature engineering techniques, and evaluating their performance on larger and diverse datasets. This project contributes to the field of cybersecurity by showcasing the potential of machine learning in malware detection and prevention.

References

[1] Idika, N., & Mathur, A. P. (2007). A survey of malware detection techniques. Purdue University, 48(2), 32-46.

[2] Dhanya, K. A., Dheesha, O. K., Gireesh Kumar, T., & Vinod, P. (2021, February). Detection of obfuscated mobile malware with machine learning and deep learning models. In Machine Learning and Metaheuristics Algorithms, and Applications: Second Symposium, SoMMA 2020, Chennai, India, October 14–17, 2020, Revised Selected Papers (pp. 221-231). Singapore: Springer Singapore.