Project Report for

**Reverse Engineering on Educational Malware**

Submitted to:

**Professor Dr. Makan Pourzandi**

**Date: 29th August 2022**

Submitted By:

**AJAY DEVJANI**

Contents

[1. Abstract 1](#_Toc114593714)

[2. Introduction 1](#_Toc114593715)

[3. Related Work 1](#_Toc114593716)

[3.1 PE File Format 1](#_Toc114593717)

[3.2 n-grams 2](#_Toc114593718)

[3.3 K-Fold Cross Validation 2](#_Toc114593719)

[3.4 scikit-learn for Machine Learning 3](#_Toc114593720)

[3.5 Machine Learning Approaches 3](#_Toc114593721)

[3.6 Comparison of various tools for Malware Analysis 4](#_Toc114593722)

[4. System Design 4](#_Toc114593723)

[4.1 Static and Dynamic Analysis 4](#_Toc114593724)

[4.2 Design and Flow 5](#_Toc114593725)

[5. Implementation 6](#_Toc114593726)

[6. Experimental Results 8](#_Toc114593727)

[7. Conclusion 10](#_Toc114593728)

[8. References 10](#_Toc114593729)

[9. Appendices 12](#_Toc114593730)

[Appendix A: Installation of IDA Free 12](#_Toc114593731)

# Abstract

Artificial intelligence (AI) and the digitization of everything, which makes the world a global hub, constitute the fourth industrial revolution. Due to the increasing use of computer and information technology, cyber attacks on sensitive data have emerged as important global dangers. Cybercriminals often publish new malware or dangerous programmes online to steal or delete crucial data. Therefore, the cyber world is very interested in studies on data protection. Machine learning techniques can be utilised for precise categorization and detection to deal with emerging dangerous software variations. In this project, we are focusing on behavioral based detection of Windows malware (exe files) by extracting API call sequences before and after executing an exe file. IDA Free was used for extracting static API sequence and Cuckoo Sandbox was used for extracting dynamic API sequence. After extracting, various machine learning algorithms with n-grams and cross validation were canvassed to determine the accuracies. The contrast of accuracies of different approaches is provided at the end.

# Introduction

Malware compromises the integrity and privacy of data and results in the unintentional disclosure of information. Cyber attacks have been rapidly rising in recent years due to new computer and internet applications. Through the Internet, cybercriminals publish tens of thousands of brand-new malwares to steal or obliterate crucial data. In consequence, the computer user community places a high priority on the effective identification and prevention of hostile insiders to safeguard sensitive data. In coordination, malware analysis serves as the foundation and initial step in the malware detection process. Its goal is to identify the characteristics of the malware that can be utilised to identify it. Behavioral based malware analysis and detection is a crucial technology that extracts the behaviour of malware before or during runtime, and produces evidence for recovery, cleanup, and forensics.

In this project, to analyze the behavior of Windows exe files, we are using IDA Free for extracting static API sequence and Cuckoo Sandbox for extracting dynamic API sequence. Installation steps of IDA Free and steps to extract API sequences are available at Appendices section. Firstly, the API sequences were split in n-grams and for cross validation into k folds with shuffling inputs. These API sequences are then canvassed to various machine learning approaches i.e., Gaussian Naïve Bayes, Support Vector Machines (SVM), Decision tree and Random Forest to get accuracies of each approach.

# Related Work

According to the journal [1], malware detection/classification using machine learning methods has been extensively researched/studied in recent years. Initially, a huge dataset of malwares and benign programs is collected and then using malware analysis tools like IDA Free/Pro or Cuckoo Sandbox, API sequences are extracted for further analysis. The API sequences are pre-processed and further on provided as inputs to the machine learning algorithms. We have performed same methodology but with different machine learning algorithms.

## PE File Format

In static analysis, one can analyse dynamically linked libraries (DLLs) and win32 application programming interfaces (APIs) of an executable to forecast its behaviour (APIs). This can be easily deduced from the file's PE format's import section (.idata section). Figure 1 depicts the PE format's main structure, while the figure besides it, depicts some of the format's imported DLLs and APIs. It's essentially a data structure that contains the information the Windows OS loader needs to manage the wrapped executable code. We can get what we require from the idata section by parsing the PE format. As a result, we can predict the behaviour of an executable programme before it is run.

Figure 1: PE File Format

A picture containing graphical user interface

Description automatically generated

## n-grams

N-grams are continuous sequences of words, symbols, or tokens in a document, where 'n' is just a variable with positive integer values such as 1, 2, 3, and so on. They can be defined as the adjacent sequences of items in a document in technical terms. In the project, we have discussed about 3-grams, 4-grams and 5-grams. In 3-grams, the API call sequence is divided into batch of 3 sequences.

Table

Description automatically generated

Figure 2: 3-grams

Similarly, in 4-grams and 5-grams, the API call sequence is divided into batch of 4 sequences and 5 sequences respectively.

## K-Fold Cross Validation

Cross validation is a statistical technique that divides the available data into a pre-set number of folds, or partitions, in a methodical manner. Classifiers are generated by applying a learning algorithm to n-1 folds and then evaluated on the nth fold. Once the evaluation of all folds is finished, the mean of cross validation is calculated. Even though criticism has been directed towards (over) belief in the cross-validation performance estimates, the method is still widely regarded as a reasonable and robust performance estimation method, especially when the data is scarce. In the project, we have evaluated 5, 10 and 20 folds with input being shuffled every time.

## scikit-learn for Machine Learning

After the n-grams are created, these n-grams text file is then given as input to machine learning algorithms. [Scikit-learn](https://scikit-learn.org/stable/index.html) is an open-source machine learning library that supports supervised and unsupervised learning. It also provides various tools for model fitting, data preprocessing, model selection, model evaluation, and many other utilities. It contains efficient tools for predictive data analysis accessible to everybody, and reusable in various contexts.

## Machine Learning Approaches

#### Gaussian Naïve Bayes

In the beginning, we use the Gaussian Naive Bayes, a probabilistic classifier that estimates the probability that an input belongs to a particular class. In contrast to other classification models, it is relatively fast since it uses binary variables, such as 1/0 or true/false, to express the presence or absence of features. Moreover, this classifier outperforms others when the conditional independence assumption for classes is met. Furthermore, it has been shown that this probabilistic technique performs well in text classification.

#### Support Vector Machines

Support vector machines (SVMs) are a set of supervised learning methods used for [classification](https://scikit-learn.org/stable/modules/svm.html#svm-classification), [regression](https://scikit-learn.org/stable/modules/svm.html#svm-regression) and [outliers detection](https://scikit-learn.org/stable/modules/svm.html#svm-outlier-detection). Finding a hyperplane in an N-dimensional space (N is the number of features) that categorises the data points clearly is the goal of the support vector machine algorithm. There are a variety of different hyperplanes that might be used to split the two classes of data points. The AIM is to find a plane with the greatest margin i.e., the greatest separation between data points from both classes. By increasing the margin distance, some support is provided, increasing the confidence with which subsequent data points can be categorised.

#### Decision Trees

Decision Trees (DT) are non-parametric supervised learning technique for classification and regression. The objective is to learn straightforward decision rules derived from the data features to build a model that predicts the value of a target variable. A decision tree can be used in decision analysis to reflect decisions and decision-making formally and visually. It employs a decision-tree-like approach, as the name suggests.

#### Random Forest

Moreover, we test random forest which is an ensemble learning method mostly used for classification. Random forests are composed of several decision trees that are combined to make a unanimous decision or classification. Random forest is better than just regular decision trees because they do not cause overfitting of the data.

## Comparison of various tools for Malware Analysis

|  |  |  |  |
| --- | --- | --- | --- |
| **Features** | **Ghidra** | **x64dbg** | **IDA Free** |
| **License** | Open Source | Open Source | Freeware and License |
| **Decompiler** | Available | Available (Plug In) | Cloud Decompiler |
| **Development** | Mature but Lot of bugs | Mature | Very Mature. Development since 1990s |
| **Binaries Support** | Load of Multiple Binaries at once | Load of Limited Binaries at once | Load of Limited Binaries at once |
| **Disassembler** | Available | Available | Available |
| **Architecture Support** | Supports limited microprocessors | Supports limited microprocessors | More than 65 families of microprocessors are supported |
| **USP** | Decompiler | Symbols | GUI and Usability |

# System Design

## Static and Dynamic Analysis

There are basically two types of malware analysis: static analysis and dynamic analysis. Static Malware Analysis (SMA) is the basic malware analysis method during which there is no need to execute the malware. Using IDA Free, we will get API Call Sequence without executing exe.

Dynamic Malware Analysis (DMA), on the other hand, allows a programme to run and monitors the execution process for any aberrant behaviours that may present. Using Cuckoo Sandbox, we will get API Call Sequence during the execution of exe.

## Design and Flow

***Data Collection***: We have collected total of 83 Malwares and 22 Benign Programs for static analysis and 83 Malwares and 31 Benign Programs for dynamic analysis. The dataset was downloaded from [MalwareBazaar](https://bazaar.abuse.ch/browse/) and [figshare](https://figshare.com/articles/dataset/Malware_Detection_PE-Based_Analysis_Using_Deep_Learning_Algorithm_Dataset/6635642).

***Data Extraction***: After the data is collected it is then provided as input to IDA Free and Cuckoo Sandbox. From Cuckoo Sandbox, CSVs of API sequence of executables during execution can be directly generated. Wherein for IDA Free, python code to convert hexadecimals to decimals followed by an IDC script is executed in order to generate a log file which contains the sequence of API calls.

***Data Pre-processing***: After data is extracted, it is then split into n-grams, in our case, 3-grams, 4-grams and 5-grams. All the n-grams of all executables whether it is malware or benign is dumped into single txt file with name "training\_<<n-gram size>>\_gram.txt". Similarly for dynamic analysis, all the n-grams generated from CSVs are dumped into single file with name "training\_dynamic\_<<n-gram size>>\_gram.txt". These txt files are having data in format: <<Bar separated API-Sequence, Is Malware (1)/Benign (0)>>. These txt files are the perfect inputs to be provided for ML methods.

Diagram

Description automatically generated

Figure 3: System Design and Flow

***Data mining/Machine Learning methods:*** After the data is pre-processed, it is then canvassed to 4 ML methods like Gaussian Naïve Bayes, SVM, Decision Tree and Random Forest. All the above four methods are executed for static and dynamic analysis with k-folds, in our case, No Fold, 5-folds, 10-folds and 20-folds. Accuracies of these methods with different folds/analysis are calculated and accuracy report will be published.

# Implementation

#### Hexadecimal Addresses to Decimal Addresses

First step after loading file in IDA Free is to convert the hexadecimal addresses of “Import” section inside IDA Free to decimal addresses. To perform this step, copy the “Import” section and paste to newly created text file. After that execute the hex2dec.py with the path of text file as a command line argument. Please find below the pseudo code for hex2dec.py.

*Open and read the file containing list of hexadecimal addresses of all API imports.*

*Convert each hexadecimal into decimal and store it in a list.*

*Write every element of list in a new file containing decimals.*

#### IDA Scripting

After converting hexadecimal to decimal addresses, we have used IDC script in IDA Free to get static API sequence.

Below are the steps to get API sequence using IDA Free:

1. Load exe file in IDA Free.
2. Copy HEXS from import section in IDA Free.
3. Convert those HEXS to decimals using hex2dec.py.
4. Run IDA Script. It’ll create log in output folder.
5. Copy that log to Python project in “inputs” folder for further analysis.
6. Please follow readme.txt for generating report.

Pseudo Code for IDC script:

*Open the file containing decimal addresses.*

*Loop until every address (of all API imports) in file is traversed:*

*Get the function name available on specified address.*

*Loop until all references of function on specified address are traversed:*

*Get the function name of reference*

*Loop until all in code functions are traversed:*

*Get the function name available on specified address.*

*Loop until all references of function on specified address are traversed:*

*Get the function name of reference*

This will generate a log file in format, <<To\_API>>, <<From\_API>>, <<Invoke Address>>, which can be further processed and canvass to ML methods.

#### Data Preprocessing

These log files are then preprocessed using log2stats.py and output is stored in “outputs” folder where all these log files are converted into respective txt file with format, <<Bar Separated API Sequence>>, 0/1 wherein 1 If Malware and 0 if Benign.

Pseudo Code:

*Traverse every file in inputs/malware folder:*

*Sort the data frame, first with “From” and then with “Address”.*

*Iterate every row in data frame:*

*Open a file in ‘outputs’ folder with same name in ‘write’ mode.*

*Write the APIs in format: 0 << Indicates it is malware>>*

*<From>|<To 1>|<To 2>|……| <To n>*

*Similarly repeat steps for every file in inputs/benign folder.*

#### n-grams creation

The txt containing entire API Sequence is then split into n-grams.

Pseudo Code:

*Traverse each file in ‘outputs’ folder:*

*Get first line. First line indicates whether it’s malware or benign.*

*Split the second line with ‘|’.*

*If length of split tokens is less than n-gram size:*

*Pad it with ‘\_\_NaN\_\_’*

*Loop every token (generated above using split):*

*Take (n-gram size – 1) tokens, join them into a string*

*Append is\_malware to the above string*

*Append the above string into n-grams list*

*Open training\_<<n-gram\_size>>.txt file in ‘write’mode:*

*Loop n-gram list:*

*Write n-gram in file.*

*Similarly, for dynamic API CSV generated by Cuckoo Sandbox is then given as input to dynamic\_csv\_txt to generate n-grams. So, repeat the above steps for dynamic analysis txt.*

#### Machine Learning Models:

Pseudo code:

*Open specific training txt file according to static or dynamic analysis.*

*Read the dataset: Format: <<API Sequence>>, <<Is Malware>>*

*Encode API sequence to provide each sequence a label.*

*Split the test and training data.*

*Define the classifier: GaussianNB, LinearSVM, Decision Tree or Random Forest*

*Fit the model into classifier.*

*Run the prediction to get accuracy and store the accuracy in list.*

*Define the folds.*

*Loop until all the folds are completed:*

*Split the dataset in k-folds with shuffling.*

*Get the accuracy of each fold and store it in accuracy list.*

# Experimental Results

Table 1 shows the static and dynamic analysis on 3-grams of various ML methods in 4 different fold types. In static analysis, we can clearly see that SVM outperforms other 3 methods in each of 4 types of folds with highest accuracy of 86.15% with 20 folds. If we talk about dynamic analysis, there is significant rise in accuracies of all ML methods with Decision Tree with exceptional rise of around 5%. The cross validation wasn’t found that effective in dynamic analysis and even sometimes degrading performance in case of Decision tree. SVM with 20 folds have shown maximum accuracy in both static and dynamic analysis with 86.15% and 88.06% respectively compared to all other combinations.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **STATIC ANALYSIS** | | | | |
|  | **No Fold** | **5-Fold** | **10-Fold** | **20-Fold** |
| **Naive Bayes** | 77.972219 | 78.135029 | 78.135028 | 78.135027 |
| **SVM** | 85.662724 | 85.780535 | 86.023655 | 86.151778 |
| **Decision Tree** | 78.455024 | 78.604084 | 78.604084 | 78.604082 |
| **Random Forest** | 77.972219 | 78.135029 | 78.135028 | 78.135027 |
|  |  |  |  |  |
| **DYNAMIC ANALYSIS** | | | | |
|  | **No Fold** | **5-Fold** | **10-Fold** | **20-Fold** |
| **Naive Bayes** | 80.708486 | 80.817899 | 80.817898 | 80.817914 |
| **SVM** | 87.864727 | 88.026021 | 88.05279 | 88.062604 |
| **Decision Tree** | 83.099848 | 82.896992 | 82.896994 | 82.896999 |
| **Random Forest** | 80.708486 | 80.817899 | 80.817898 | 80.817914 |

Table 1: Static and Dynamic Analysis on 3-grams

Table 2 shows the static and dynamic analysis on 4-grams of various ML methods in 4 different fold types. In static analysis, we can clearly see that SVM outperforms other 3 methods in each of 4 types of folds with highest accuracy of 84.42% with 20 folds. If we talk about dynamic analysis, the cross validation wasn’t found that effective in dynamic analysis again, but it didn’t degraded performance of any ML method. SVM with 20 folds have shown maximum accuracy in both static and dynamic analysis with 84.42% and 88.89% respectively compared to all other combinations.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **STATIC ANALYSIS** | | | | |
|  | **No Fold** | **5-Fold** | **10-Fold** | **20-Fold** |
| **Naive Bayes** | 78.176194 | 78.150964 | 78.150964 | 78.150964 |
| **SVM** | 84.151863 | 84.09933 | 84.320523 | 84.427317 |
| **Decision Tree** | 78.501071 | 78.49174 | 78.497961 | 78.505564 |
| **Random Forest** | 78.176194 | 78.150964 | 78.150964 | 78.150964 |
|  |  |  |  |  |
| **DYNAMIC ANALYSIS** | | | | |
|  | **No Fold** | **5-Fold** | **10-Fold** | **20-Fold** |
| **Naive Bayes** | 80.712339 | 80.840361 | 80.840363 | 80.840365 |
| **SVM** | 88.532869 | 88.88998 | 88.890876 | 88.889984 |
| **Decision Tree** | 82.647361 | 82.682493 | 82.682496 | 82.682498 |
| **Random Forest** | 80.712339 | 80.840361 | 80.840363 | 80.840365 |

Table 2: Static and Dynamic Analysis on 4-grams

Table 3 shows the static and dynamic analysis on 5-grams of various ML methods in 4 different fold types. In static analysis, we can clearly see that SVM outperforms other 3 methods in each of 4 types of folds with highest accuracy of 83.40% with 20 folds. If we talk about dynamic analysis, the cross validation wasn’t found that effective in dynamic analysis again, but it degraded performance sometimes. SVM with 20 folds have shown maximum accuracy in static analysis with 83.40% and in dynamic analysis with 10 folds and 89.307% accuracy compared to all other combinations.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **STATIC ANALYSIS** | | | | |
|  | **No Fold** | **5-Fold** | **10-Fold** | **20-Fold** |
| **Naive Bayes** | 77.972592 | 78.177586 | 78.177586 | 78.177582 |
| **SVM** | 82.900319 | 83.097804 | 83.307302 | 83.403039 |
| **Decision Tree** | 78.13028 | 78.31162 | 78.31162 | 78.305984 |
| **Random Forest** | 77.972592 | 78.177586 | 78.177586 | 78.177582 |
|  |  |  |  |  |
|  |  |  |  |  |
| **DYNAMIC ANALYSIS** | | | | |
|  | **No Fold** | **5-Fold** | **10-Fold** | **20-Fold** |
| **Naive Bayes** | 80.684006 | 80.862894 | 80.862902 | 80.862894 |
| **SVM** | 88.831192 | 89.275163 | 89.307394 | 89.304709 |
| **Decision Tree** | 82.523837 | 82.645444 | 82.645446 | 82.64544 |
| **Random Forest** | 80.684006 | 80.862894 | 80.862902 | 80.862894 |

Table 3: Static and Dynamic Analysis on 5-grams

Static analysis will have maximum accuracy of 86.15% if performed with SVM having 3-grams and 20 folds. On the other hand, dynamic analysis will have maximum accuracy of 89.307% if performed with SVM having 5-grams and 10 folds.

# Conclusion

We chose IDA Free because it was easy, free, and reliable tool to find static API sequence. We chose Cuckoo Sandbox for extracting dynamic API sequence because it executes exe in a standalone isolated environment which makes the Cuckoo Sandbox an extremely safe tool. These API sequences were initially preprocessed and were then split in n-grams along with k folds with shuffling inputs. These API sequences are then canvassed to various machine learning approaches i.e., Gaussian Naïve Bayes, Support Vector Machines (SVM), Decision tree and Random Forest to get accuracies of each approach. Experimental results show SVM outperformed other three in both static and dynamic analysis. SVM performed on static API sequence with 3-grams and 20 folds shown 86.15% accuracy whereas on dynamic API sequence with 5-grams and 10 folds shown 89.307% accuracy.

# References

[1] R. M. A. Molina, S. Torabi, K. Sarieddine, E. Bou-Harb, N. Bouguila and C. Assi, "On Ransomware Family Attribution Using Pre-Attack Paranoia Activities," in IEEE Transactions on Network and Service Management, vol. 19, no. 1, pp. 19-36, March 2022, doi: 10.1109/TNSM.2021.3112056.

[2] F. Mira, "A Futuristic Appraisal of Malware Detection by Employing Data Mining," 2022 5th International Conference on Computing and Informatics (ICCI), 2022, pp. 204-208, doi: 10.1109/ICCI54321.2022.9756097.

[3] W. Liu, P. Ren, K. Liu and H. -x. Duan, "Behavior-Based Malware Analysis and Detection," 2011 First International Workshop on Complexity and Data Mining, 2011, pp. 39-42, doi: 10.1109/IWCDM.2011.17.

[4] Tzu-Yen Wang, Shi-Jinn Horng, Ming-Yang Su, Chin-Hsiung Wu, Peng-Chu Wang and Wei-Zen Su, "A Surveillance Spyware Detection System Based on Data Mining Methods," 2006 IEEE International Conference on Evolutionary Computation, 2006, pp. 3236-3241, doi: 10.1109/CEC.2006.1688720.

[5] R. K. Shahzad, S. I. Haider and N. Lavesson, "Detection of Spyware by Mining Executable Files," 2010 International Conference on Availability, Reliability and Security, 2010, pp. 295-302, doi: 10.1109/ARES.2010.105.

[6] M. Chowdhury, A. Rahman and R. Islam, "Protecting data from malware threats using machine learning technique," 2017 12th IEEE Conference on Industrial Electronics and Applications (ICIEA), 2017, pp. 1691-1694, doi: 10.1109/ICIEA.2017.8283111.

# Appendices

## Appendix A: Installation of IDA Free

The Interactive Disassembler (IDA) as a disassembler can create maps of their execution to show the binary instructions that are actually executed by the processor in a symbolic representation (assembly language). Advanced techniques have been implemented into IDA Pro so that it can generate assembly language source code from machine-executable code and make this complex code more human-readable. The debugging feature augmented IDA with the dynamic analysis. It supports multiple debugging targets and can handle remote applications. Its cross-platform debugging capability enables instant debugging, easy connection to both local and remote processes and support for 64-bit systems and new connection possibilities.

Freeware of IDA i.e., IDA Free can be downloaded from <https://hex-rays.com/ida-free/#download>

**IDA Free comes with basic features and doesn’t support the IDAPython and other advanced features.**

**Important Note:** For the purpose of safe and secure environment, the experiments were performed inside Oracle Virtual Box (Windows 7), so that all the malware exes are in isolated environment. We encourage to follow the same process of collecting and storing the malwares in isolated environment for static analysis using IDA.