MALWARE ANALYSIS TOOL

Bhuvanesh V T[1], Mukesh N[2], Praveen S[3], Lakshmanaprakash S[4]

[1] *Student, Information Technology, Bannari Amman Institute of Technology, Tamil Nadu, India*
[2] *Student, Information Technology, Bannari Amman Institute of Technology, Tamil Nadu, India*
[3] *Student, Information Technology, Bannari Amman Institute of Technology, Tamil Nadu, India*
[4] *Assistant Professor, Information Technology, Bannari Amman Institute of Technology, Tamil Nadu, India*

ABSTRACT

The proliferation of malware threats in the digital landscape has posed significant challenges to cybersecurity practitioners. In response to this evolving threat landscape, this research paper explores a holistic approach to malware analysis that leverages both reverse engineering and machine learning techniques. Reverse engineering allows for the in-depth examination of malicious code, enabling a deeper understanding of its functionality and evasion techniques. Concurrently, machine learning models are employed to automate the classification and detection of malware based on extracted features and behavioral patterns. This research investigates various aspects of malware analysis, including the reverse engineering of malware binaries to uncover their inner workings, identification of malicious code patterns, and extraction of relevant features. Additionally, machine learning algorithms are trained on these features to distinguish between benign and malicious software efficiently. Furthermore, this paper discusses the advantages and challenges of combining reverse engineering and machine learning in the context of malware analysis. It highlights the benefits of increased accuracy and scalability in detecting emerging malware threats, as well as the potential pitfalls and limitations of the approach. Moreover, the paper explores the significance of continuously updating machine learning models to adapt to evolving malware tactics. The experimental results presented in this study demonstrate the effectiveness of the proposed methodology in detecting previously unknown malware samples and improving overall threat intelligence. By fusing reverse engineering expertise with machine learning capabilities, this research offers a promising avenue for enhancing the resilience of cybersecurity systems against ever-evolving malware threats.

Keyword: *Malware analysis, Reverse engineering, Machine learning, Malware detection, Dimensionality reduction, Malware classification, Cybersecurity.*

1. INTRODUCTION

Introduction Malware analysis tools play a key role in cybersecurity by providing the necessary means to dissect and understand malicious software, commonly known as malware. Malware is a broad term that includes various types of malicious software such as viruses, worms, trojans, ransomware, spyware, and more. These digital threats pose significant risks to individuals, organizations and even nations as they can lead to data breaches, financial losses and disruption of critical services. Malware analysis is the process of examining and understanding the inner workings of malware to uncover its intent, capabilities, and potential impact. Malware analysts use specialized tools to reverse engineer malware, extract valuable information, and design effective countermeasures to protect systems from these threats.

1.1 Importance of Malware Analysis Tools

Introduction Threat detection and identification: Malware analysis tools help identify and categorize different types of malware, allowing cybersecurity professionals to quickly spot new and emerging threats. Behavioral analysis: These tools help analyze the behavior of malware in controlled environments and determine how it interacts with the operating system and other software components. Signature generation: Malware analysts create unique signatures based on specific malware characteristics, which antivirus software then uses to detect and block malware on infected systems. Forensic and incident response: By analyzing malware, analysts can reconstruct the attack chain and gather evidence for incident response and digital forensics. Malware family classification: Malware analysis tools help group similar malware samples into families and help researchers understand the origins and evolution of different strains.

1.2 Types of Malware Analysis Tools

- Static Analysis Tools: These tools examine the code and structure of malware without actually running it. They identify patterns, signatures and indicators of compromise (IOCs) to help classify malware and develop detection rules.

- Dynamic Analysis Tools: These tools run malware in a controlled environment, such as a sandbox, to monitor its behavior, network interactions, and system modifications. This approach helps uncover hidden malicious activity.

- Behavioral Analysis Tools: These tools focus on real-time monitoring of malware to analyze its actions, resource usage, and potential damage. They provide insight into how the malware interacts with the target system.

- Memory Analysis Tools: These tools help extract valuable information from system memory, uncover hidden processes, embedded code, and other advanced malware persistence techniques.

- Decompilers: Decompilers convert compiled malware binaries back into source code, facilitating deeper analysis and understanding of malware functionality. Malicious software, or malware, poses a significant threat to both businesses and individuals, as it can engage in activities like intercepting network data, encrypting or deleting files, and, in extreme cases, causing severe hardware failures by infecting or replacing existing firmware. The symptoms of a malware infection can vary widely based on the specific type or family of malware involved. These varying symptoms correspond to different levels of risk for infected systems. Categorizing malware helps us to analyze the potential threat posed by a particular file.

2. LITERATURE REVIEW

This research contains various methods to observe attributes of the malware. The majority of antivirus software checks a file's syntactic signatures to see if it contains any dangerous code. These signatures might include combinations or patterns of previously known harmful instruction combinations or patterns. Antivirus software can examine a file's hash in more detail and compare it to a database of hash values from samples of known malware. The drawback of the signature-based strategy employed by the majority of antivirus products is that minor modifications to the malware's source code may produce a signature that hasn't been seen before and, as a result, isn't listed in the most recent database of signature definitions. Similar to that, these signature-based techniques offer scant defense against zero-day assaults. Malware that is metamorphic or polymorphic might have numerous signatures, which complicates signature-based detection methods. To counter the aforementioned syntactic signature-based evasion techniques, some research has been done on semantics-based detection systems. These systems use abstractions and templates to try to determine what the executable does rather than how it accomplishes it. In order to successfully detect, analyze, and mitigate these threats, powerful malware analysis tools are now essential as the sophistication and frequency of malware attacks continue to rise. This research provides a comprehensive understanding of malware analysis, its goals, and the different types of analysis techniques (static, dynamic, and hybrid).

2.1 Dynamic Analysis

Dynamic analysis entails executing malware within a controlled environment, typically on a virtual machine. During this process, the behavior of the malware while running is closely observed, with a focus on monitoring system events and network activity. This approach to malware analysis offers a significant advantage by reducing the uncertainty that static analysis may have, as the malware operates in an environment that closely resembles actual system conditions. However, dynamic analysis is not without its drawbacks. There is a nonzero risk of the malware spreading to other machines on the same network during the analysis. Additionally, researchers have encountered instances where malware exhibits different behavior when running within a virtualized environment, making it challenging to achieve accurate categorization.

2.2 Static Analysis

Static analysis is a method employed to assess the characteristics of malware without the need to load its executable file into memory. This approach revolves around scrutinizing the malware file's heuristics for indicators that provide insights into potential behaviors it might exhibit if executed. One clear advantage of static analysis is that it eliminates the risk of the malware spreading to other networked machines and safeguards the host operating system from harm since the malware remains dormant. However, static analysis does come with its limitations as referred with. It is unable to fully dissect the behavior of a binary that employs techniques like self-modifying code or relies on dynamic data such as the current date and time. Achieving precise results through static analysis can be computationally demanding, posing challenges, especially for systems requiring real-time threat detection. It's worth noting that not all static analysis systems face this issue; for instance, the PE Miner tool demonstrated near-real-time detection capabilities.

3. METHODOLOGY

The goal of developing a malware analysis tool is to create a cutting-edge solution that enables cybersecurity professionals to effectively identify, classify, and understand malicious software, thereby enhancing overall security preparedness and enabling rapid responses to emerging threats. This comprehensive tool will include dynamic analysis that examines malware in controlled environments to reveal its behavioral characteristics, static analysis to examine code without execution, behavioral analysis to track and record malicious actions, signature-based detection, and innovative heuristic and machine learning techniques. Detection of previously unknown malware. It will provide clear visualization and reporting of analysis results, seamless integration with threat intelligence feeds, ensure scalability and optimal performance, offer an intuitive user interface for simplified use, and prioritize continuous updates and research to stay at the forefront of the evolving threat landscape. Achieving this goal will make the malware analysis tool the cornerstone of an organization's cybersecurity strategy, facilitating rapid threat detection and response.

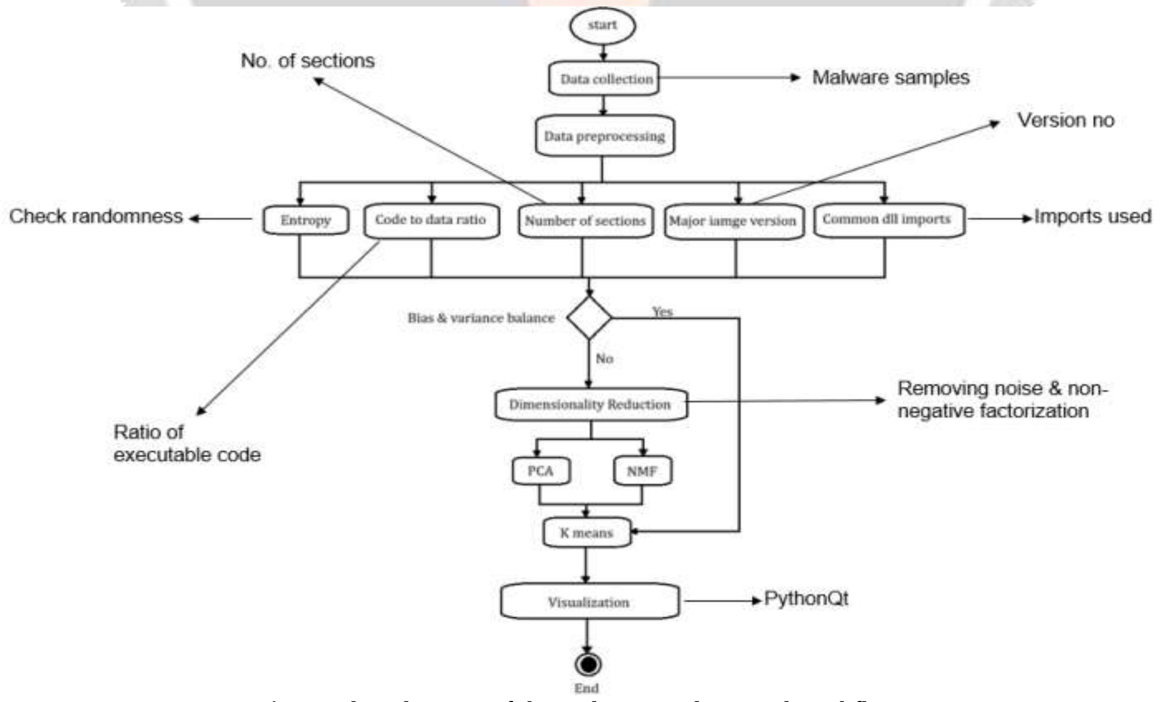

Fig -1: Flow diagram of the malware analysis tool workflow

3.1 Calculation of Entropy

An empty array of 256-bit integers of length zero is first initialized before doing the entropy computation. You may determine the entropy of individual file subsections as well as the overall entropy value using the calculateEntropy method. The count index in the array is also a decimal value for that particular byte, allowing for efficient iteration of the array afterwards. This array will store the number of occurrences of a specific byte value between startOffset and startOffset+length. Then, after looping through the byteOccurrences array, any byte values that did not occur are disregarded. We determine the frequency for every byte value that has happened, and then we set the entropy value to be equal to: prior entropy value - frequency * base 256 log frequency * 8. An entropy value between 0 and 8 is the outcome.

3.2 Ratio of Code to Initialized Data

This is the Windows 32-bit file's startup file to executable code ratio. When we first looked at data mining from both good and bad data, we discovered that good data frequently contained a lot of enormous data but just a small amount of successful code. Instead, I've discovered that faulty data frequently contains little to no data, yet the majority of the code still works. We choose to treat this example as a single machine learning job rather than two distinct tasks involving big numbers and initial knowledge in order to analyze these insights and make them more relevant.

3.3 Versions of the Main Image

This performance indicator is positive. According to Raman, benign profiles typically have a larger value than malignant profiles, which typically have a value of zero. In my initial study, we noticed the similar pattern; this is the easiest way to tell if the data we come across is unreliable or malicious.

3.4 Number of Sections

The chapter title table has this many chapter names. The number of sections in a file is an excellent indicator of whether the file is good or terrible, In general, normal files have 0 to 10 sections, but malicious files typically have 3 or 4 sections. When we look at the dataset, We can see this pattern as well.

3.5 Common DLL Imports

The DLL that the executable imported provides a very thorough description of the function and motive of the executable. For instance, you can infer that a program is utilizing the network if it importseWsock32.dll. Despite the possibility of overlap in the import of malicious and malicious files, it should be possible to categorize malware by family using this information because families behave differently. We advise setting the flag to true or false when looking for a certain import in order to obtain a number that corresponds to the item. The table displays the dll file's functionalities.

3.6 Byte to Decimal Data Conversion

It was crucial to be able to translate between the hexadecimal byte strings extracted from the 32-bit file and their equivalent decimal representations since the data required for decomposition and clustering had to be in numerical format. Using Python's built-in capability to convert between numbers in multiple base representations, the straightforward approach above was developed to avoid repeating this conversion procedure.

3.7 Reading Values from 32-bit File

It was crucial to build these methods in a way that would make them as reusable as feasible due to the tool's frequent need to read values from a file. The readByte method is the Portal Executable 32 class's smallest component, and each time the class is used to retrieve a value, one or more calls to the readByte method are made. This is demonstrated by the readBytesmethod, which cycles over the given number of bytes until it has finished reading them, attaching each one to a local variable that will be returned to the caller. The bigEndian2 bytes and read littleEndian bytes methods extend the readBytesmethod with a new level of abstraction; the only difference in the output is that the read bigEndian bytes function reverses the order of the characters in the returned byte string using Python's slicing function. The Windows Portal Executable 32-bit file specification does not guarantee that a value can be found at an offset from the beginning of the file, only that it can be found relative to a previously calculated offset, typically a COFF header, an optional header, or a Portal Executable header. It should also be noted that these two methods accept parent offsets as well as offsets that are relative to the parent offset.

3.8 Feature Extraction

By creating an instance of the Portable Executable 32 class with each valid file in the specified path, features are extracted. The only files for which the program was created are.EXE and.dll files, therefore it will always attempt to extract functionality from those. The Portable Executable 32 class's methods are used to retrieve the values from the file after a Portable Executable 32 class instance has been created. The associated filenames are then recorded in a another csv file called files.csv, whereas these values are stored in a csv file called features.csv. Because further steps of the process used by the finished tool require that the data input into the algorithms be numeric, it is necessary to save the file names separately. We only read from files when filenames are required.csv, and we only read from features when we need information for our machine learning algorithms. The initial part of the data in one file will correlate with the first row of data in the second, etc., because the indexes in csv files are consistent.

3.9 User Interface

Two classes, UIMainWindowx and Portable Executable Machine Learning, are used to construct the user interface for the finished tool. Portable Executable Machine Learning utilizes UIMainWindow as a superclassx, inheriting all of the superclass's attributes. The layout, nomenclature, and the majority of the display details of the employed UI components are handled by UIMainWindow, which serves as a UI setting and cannot be constructed without being inherited. In order to display the outcomes of these actions on the user interface, the PortableExecutable MachineLearning class uses its superclass connection to handle additional specific activities like plotting graphs. Additionally, the PortableExecutableMachineLearning class makes use of the PortableExecutableUtil class to carry out a few non UI/UX based tasks like computing precision values and carrying out IO operations.

3.10 Clustering and Categorization

The PortableExecutableMachineLearning class handles clustering and categorization. Methods can accept parameters to adjust their behavior and outputs when necessary and have been designed to be flexible and reusable.

4. RESULTS AND DISCUSSIONS

The results are generated from extracting features from a Windows32 PE (Portable Executable) file, it can involve various techniques to gain insights into the file's characteristics Here are some of the features involved:

- Entropy of Sections
- Number of Sections
- Code to Data Ratio
- Major Image Versions
- Common DLL Imports

4.1 Dimensionality Reduction Techniques

Dimensionality reduction techniques are methods used in data analysis and machine learning to reduce the number of variables or features in a dataset while preserving essential information. These techniques are used for various reasons:

4.2 Visualization

Reducing the dimensionality of data allows for easier visualization of complex datasets. By projecting data into a lower-dimensional space (e.g., 2D or 3D), patterns and relationships between data points can become more apparent.

4.3 KMeans

After performing the Dimensionality Reduction Techniques, The data is categorized using the KMeans. K-Means is a commonly used clustering algorithm that can be applied to data with a lower number of dimensions to discover patterns and group similar data points. Here's how you can use K-Means for clustering after dimensionality reduction. K-Means aims to partition a dataset into K clusters, where K is a predefined number chosen by the user. The goal is to group data points into clusters such that each point belongs to the cluster with the nearest mean (centroid).

4.4 Principle Component Analysis (PCA)

By dividing a matrix (dataset) into two submatrices, one of which is the principal component of the structure that best represents the data in the original matrix, principal component analysis, which normally tries to minimize dimensionality, does this. Information that needs to be prepared is viewed as noise.

4.5 Negative Matrix Factorization (NMF)

Similar to fundamental factorization, negative matrix factorization entails dividing a large matrix into two smaller matrices. This approach differs from others in that it only uses a small number of data points to make the data set or matrix non-negative. This restriction means that
NMF might alter the original matrix.

4.6 Clustering Algorithms

Clustering techniques are used to find groups in data sets that may not be immediately apparent to human readers. Different access parameters and integration algorithms come in a variety of forms. The hierarchical clustering techniques KMeans and MeanShift are both evaluated in this research. Clustering techniques are used to find groups in data sets that may not be immediately apparent to human readers. Different access parameters and integration algorithms come in a variety of forms. The hierarchical clustering techniques KMeans and MeanShift are both evaluated in this research.

5. CONCLUSION

As part of this experiment, a tool was developed that can run multiple variations of separation and integration algorithms and present the results to the user. The tool also allows users to view 2D data in various formats. In this project, a system was developed that uses heuristic information obtained from Windows32 portable executables to effectively classify malware. Given the performance of existing systems, pipelines have proven to be very effective in classifying malicious or malicious files using portable executable heuristics. When used with the NMF parsing algorithm and 3 features as input to the 11clusterxKMeans clustering algorithm, the system achieved 100% accuracy. This is a significant achievement and demonstrates the possibility of using heuristic information from Windows32 portable executables to effectively classify malware. However, the system is not successful in distributing bad information to families. The accuracy of the group is only 46.76%, which is not high enough for this use. This fact may be improved with further research and testing. However, there is currently not enough evidence to use on this subject. Overall, the proposed method demonstrates the ability to identify malware using heuristic information obtained from WIN32 portable executables. Although there is room for improvement in the classification of negative data, the system has demonstrated its potential and needs further research and development.

6. REFERENCES

[1] Muhammad Shoaib Akhtar, Tao Feng, "Malware Analysis and Detection Using Machine Learning Algorithms", 2022, Symmetry 2022, 14(11), 2304;

[2] Lei Fang, Hongbin Wu, Kexiang Qian, Wenhui Wang, Longxi Han, "A Comprehensive

Analysis of DDoS attacks based on DNS" - iopscience, 2021

[3] Arkajit Datta, Kakelli Anil Kumar, Aju. D, "An Emerging Malware Analysis Techniques and Tools: A Comparative Analysis", 2021

[4] Sanfeng Zhang, Jiahao Wu, Mengzhe Zhang, andWang Yang, "Dynamic Malware Analysis Based on API Sequence Semantic Fusion", 2023, Appl. Sci. 2023, 13(11), 6526;