Memory-Based Explainable Obfuscated Malware Detector

Enroll. No.(s) – 21103103 , 21103106 , 21103122

Name of Student(s) – Mayank singh , Harsh Raj Singh, Hammad Javed

Name of supervisor – Ms.Sangeeta Mittal

Nov-2023

Submitted in Partial Fulfillment Of The Degree Of

Bachelor of Technology

In

Computer Science Engineering

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING & INFORMATION TECHNOLOGY

JAYPEE INSTITUTE OF INFORMATION TECHNOLOGY

● Table of Content

Content

DECLARATION

We hereby declare that this submission is our own work and that,to the best of our knowledge and belief ,it contains no material previously published or written by another person nor material which has been accepted for the award of any other degree or diploma of the university or other institute of higher learning, except where due acknowledgment has been made in the text.

Place:	Signature:
Date:	Name:
	Enrollment No.

<u>CERTIFICATE</u>

This is to certify that the work titled "***Memory-Based Explainable Obfuscated Malware Detector***" submitted by "Mayank singh 21103103 , Harsh Raj Singh 21103122 , Hammad Javed 21103122" in partial fulfillment for the fulfillment for the award of degree of B.Tech of Jaypee Institute of Information Technology, Noida has been carried out under my supervision. This work has not been submitted partially or wholly to any other University or Institute for the award of this or any other degree or diploma.

Signature of Supervisor	…………………………
Name of Supervisor	…………………………
Designation	…………………………
Date	…………………………

Acknowledgement

Firstly, I express my heartiest thanks and gratefulness to almighty God for his divine blessing and making it possible to complete the project work successfully.

I am grateful and wish my profound gratitude to Supervisor **Dr. Sangeeta Mittal**, Associate Professor, Department of CSE Jaypee University of Information Technology, Noida. Deep Knowledge & keen interest of my supervisor in the field of "**Memory-Based Explainable Obfuscated Malware Detector**" to carry out this project. His endless patience, scholarly guidance, continual encouragement, constant and energetic supervision, constructive criticism, valuable advice, reading many inferior drafts and correcting them at all stages have made it possible to complete this project.

I would like to express my heartiest gratitude to **Dr.Sangeeta Mittal,** Department of CSE, for his kind help in finishing my project.

I would also generously welcome each one of those individuals who have helped me straightforwardly or in a roundabout way in making this project a win. In this unique situation, I might want to thank the various staff individuals, both educating and non-instructing, who have developed their convenient help and facilitated my undertaking.

Finally, I must acknowledge with due respect the constant support and patience of my parents.

Signature of the Student	………………………..
Name of the Student	………………………..
Enrollment Number	………………………..
Date	………………………..

Signature of the Student	………………………..
Name of the Student	………………………..
Enrollment Number	………………………..
Date	………………………..

Signature of the Student	………………………..
Name of the Student	………………………..
Enrollment Number	………………………..
Date	………………………..

SUMMARY

In response to the persistent challenges posed by obfuscated malware in the cybersecurity landscape, this project aims to bridge the gap between traditional malware detection methods and cutting-edge technologies. Recognizing the critical role of cybersecurity in safeguarding digital ecosystems, the focus is on developing a Memory-Based Explainable Obfuscated-Malware Detector using advanced techniques.

The project addresses the inadequacies in current malware detection systems by integrating memory analysis, explainable AI, and machine learning. Obfuscated malware poses a significant threat due to its ability to conceal malicious intent through various techniques such as code obfuscation and API renaming. This detector strives to enhance understanding and transparency in the identification process by incorporating explainable AI methodologies.

Key components include the collection of diverse malware samples, capturing memory snapshots during execution, and extracting relevant features from these snapshots. Techniques for obfuscation detection are implemented, and machine learning models, including Random Forest, Gradient Boosting, or neural networks, are trained on labeled datasets comprising obfuscated and non-obfuscated samples. The integration of explainability with memory analysis ensures that security analysts can comprehend the decision-making process.

The project's significance lies in its potential to revolutionize malware detection, providing a more sophisticated and transparent approach to identifying obfuscated threats. The user interface, developed using the Python library Streamlit, facilitates user-friendly interactions, while the backend logic, implemented through Python, enables real-time or near-real-time predictions.

The effectiveness of the system is validated through rigorous testing, including validation on separate datasets, testing on new malware samples, and continuous improvement based on emerging threats. Ethical considerations are paramount, and the project aligns with privacy guidelines in handling malware samples and data.

In conclusion, this project represents a crucial advancement in the realm of cybersecurity, offering a solution that combines memory-based analysis, explainable AI, and machine learning to combat the challenges posed by obfuscated malware. By providing transparency in the detection process, the system contributes to a more secure digital landscape, aligning with the ever-evolving nature of cyber threats.

CHAPTER-1 INTRODUCTION

1.1 General Introduction

In the ever-evolving landscape of cybersecurity, the relentless growth of obfuscated malware poses a formidable threat. From the dawn of digital engagement, the detection and mitigation of malicious software have become paramount for ensuring the security and integrity of digital ecosystems. In the contemporary technological era, where our reliance on interconnected systems is unprecedented, the need for a robust and sophisticated approach to malware detection is more critical than ever.

The Memory-Based Explainable Obfuscated Malware Detector project represents a groundbreaking initiative to tackle the complexities of modern cyber threats. As technology advances, so does the sophistication of malware, often concealing its true intent through obfuscation techniques. Understanding and interpreting the behavior of such malware is vital for effective detection and response.

In this project, we delve into the intricate realm of memory-based malware, employing cutting-edge machine learning techniques for detection and explanation. The conventional methods of malware detection often fall short when faced with obfuscated variants. Hence, the integration of memory-based analysis provides a novel dimension to the cybersecurity landscape, enabling the identification of obscured malicious activities within a computer's memory.

The significance of explainability in malware detection cannot be overstated. As cyber threats become more intricate, understanding the rationale behind a detection becomes crucial for cybersecurity professionals. The Memory-Based Explainable Obfuscated Malware Detector not only identifies the presence of obfuscated malware but also unravels the complexities, providing clear insights into its behavior.

In a world where digital threats continue to evolve, this project stands as a testament to innovation in cybersecurity practices. By leveraging memory-based analysis and explainable AI, the detector goes beyond conventional approaches, empowering security analysts with the transparency needed to comprehend and respond effectively to obfuscated malware.

As we navigate through the digital age, safeguarding against cyber threats becomes a collective responsibility. The Memory-Based Explainable Obfuscated Malware Detector emerges as a vital tool in

fortifying our digital infrastructure, aligning with the principles of transparency and resilience in the face of an ever-changing cyber landscape.
As businesses and individuals increasingly depend on interconnected technologies, the potential impact of cyber threats has never been higher. This project not only contributes to the immediate goal of effective malware detection but also sets a precedent for future developments in explainable AI within the cybersecurity domain. The Memory-Based Explainable Obfuscated Malware Detector thus stands at the forefront of efforts to secure our digital ecosystems with intelligence, transparency, and resilience.

1.2 Problem Statement

In the ever-evolving realm of cybersecurity, the rise of obfuscated malware presents a formidable challenge, demanding cutting-edge detection solutions. Traditional approaches often fall short in identifying obscured malicious activities within a computer's memory, leading to potential security breaches and data compromise. Conventional methods frequently prove inadequate in uncovering covert malicious behaviours entrenched within a computer's memory, creating vulnerabilities that can result in security breaches and compromised data integrity. In response to this pressing challenge, the mission is to craft a sophisticated Memory-Based Explainable Obfuscated Malware Detector employing state-of-the-art machine learning methodologies.

1.3 Significance of the Problem

- Advanced Threat Landscape: With the continuous evolution of cyber threats, the prevalence of obfuscated malware poses a sophisticated and persistent danger. Developing a detector tailored for memory-based obfuscated malware is crucial to stay ahead of adversaries employing increasingly complex evasion techniques.
- Transparent Decision-Making: The incorporation of explainable AI in malware detection is essential for cybersecurity professionals to understand the rationale behind the system's decisions. Transparent and interpretable insights enable quicker and more accurate responses to potential threats, fostering a proactive cybersecurity posture.
- Adaptability to Modern Threats: As cyber threats become more sophisticated and dynamic, solutions that specifically address memory-based obfuscated malware contribute to the adaptability of cybersecurity measures. This adaptability is crucial in defending against emerging threats that may bypass conventional security protocols.
- Strategic Impact on Cybersecurity: The development of an effective Memory-Based Explainable Obfuscated Malware Detector represents a strategic advancement in cybersecurity

capabilities. It aligns with the broader goal of fortifying digital infrastructures against evolving cyber threats, thereby enhancing the overall resilience of organizations and networks.

1.4 Empirical Study

The proposed Windows malware detection system was designed to meet the following goals:

1. High accuracy: Given the rising number of obfuscated-malware attacks, the system aimed to achieve a high level of accuracy in detecting such malware.

2. Lightweight :The system was designed to be lightweight, minimizing resource consumption and processing requirements. By reducing the number of features needed for detection, it aimed to optimize performance and efficiency.

3. Explainability :The selected features and detection mechanisms of the system were designed to be explainable. This means that the system's decisions and classifications were based on transparent and interpretable conditions.

Explainability of AI Models:

- Assess the effectiveness of the system's explainable AI features. Participants will be asked to interpret and understand the factors influencing the detector's decisions, contributing to transparency and trust in the detection process.

Technological Barriers:

- Explore any challenges faced by cybersecurity professionals in implementing or utilizing the Memory-Based Explainable Obfuscated Malware Detector. Gather feedback on system integration, compatibility, and any perceived technological hurdles.

Analysis:

- Analyze the collected data to identify common trends, challenges, and success stories. Insights will be provided to highlight areas for improvement, potential enhancements, and the overall effectiveness of the Malware Detector.

Conclusion and Recommendations:

- Based on the empirical study, offer recommendations for refining and optimizing the Memory-Based Explainable Obfuscated Malware Detector. Emphasize positive impacts observed and suggest specific adjustments and improvements for enhanced cybersecurity capabilities.

Acknowledgments:

- Express gratitude to the participants, including cybersecurity professionals, experts, and project team members, for their valuable contributions to the empirical study.

1.5 Brief Description of the Solution Approach

Recommend testing multiple algorithms within the Memory-Based Explainable Obfuscated Malware Detector. By examining the classification report, compare the algorithms to select the most effective one.

Steps in the Process:

- Define the problem: Clearly outline the challenges and objectives of detecting memory-based obfuscated malware.
- Prepare the data: Curate and preprocess datasets to ensure they accurately represent diverse malware scenarios.
- Compare algorithms: Evaluate the performance of various machine learning algorithms in detecting obfuscated malware patterns.
- Find the best algorithm: Based on the comparison, choose the algorithm that demonstrates superior accuracy, precision, and recall for memory-based obfuscated malware detection.

CHAPTER-2
LITERATURE SURVEY

2.1 Summary of papers studied

There is an increasing focus on Memory-Based Explainable Obfuscated-Malware Detectors, with several studies and papers exploring various aspects of this field. The following is a summary of some of the key literature in this area.

2.1.1) Literature Survey 1

In 2018, Ali and Soomro proposed an obfuscated-malware detection system that employed particle swarm intelligence optimization (PSO) for feature selection and a random forest (RF) classifier (Ali and Soomro, 2018). The proposed system extracts features from packed and non-packed portable executable (PE) files. The main contribution of their research lies in the use of the PSO approach for feature selection. This approach prioritizes the removal of redundant, irrelevant, and noisy features from the extracted set. It is worth noting that the dataset used in their experiments was initially introduced in 2008, although the research itself was published in 2018. Consequently, the dataset did not encompass samples from the most recent malware families. When tested, the proposed system achieved an accuracy of 0.9960 by leveraging packed PE features.

2.1.2) Literature Survey 2

In 2019, Li et al. introduced a machine learning–based malware detection system called Obfusifier (Li et al., 2019). This system was specifically designed to maintain high accuracy in detecting obfuscated malware by utilizing obfuscation-resistant features extracted from unobfuscated applications and represented in method graphs. The method graphs were used to capture the calling relationships between different methods and subroutines in the malware structure. Experimental testing of the proposed system demonstrated an accuracy of 0.95 in detecting obfuscated malware when employing an RF classifier.

2.1.3) Literature Survey 3

In 2020, Park et al. introduced an obfuscated-malware detection system specifically designed for low-power devices, such as Internetof-Things devices (Park et al., 2020). The proposed method used the extraction of features from Markov matrices constructed from opcode traces. These features were then used to train a machine learning–based classifier. Experimental testing demonstrated that the proposed system achieved an accuracy of 0.926 in detecting non-obfuscated malware and an accuracy of 0.9318 in detecting obfuscated malware. However, the proposed system was proved to consume less power when compared with other methods.

2.1.4) Literature Survey 4

In 2020, Jahromi et al. introduced an improved stacked long-term short-memory (LSTM) based method for malware detection in safetyand time-critical systems (Jahromi et al., 2020). The proposed method presented an enhanced version of stacked LSTM that surpassed the performance of the original stacked LSTM in terms of accuracy and efficiency. The system was evaluated on multiple datasets, including various Windows malware datasets, and achieved accuracy levels ranging from 80% to 100%. However, it is important to note that the Windows malware datasets used in the study had a relatively small number of samples compared to the dataset utilized in the current research. Furthermore, these datasets did not specifically address the detection of obfuscated malware.

2.1.5) Literature Survey 5

Darabian et al. presented, in 2020, a deep learning–based approach for detecting malware that combined static and dynamic features (Darabian et al., 2020). While the study specifically focused on detcryptomining malware, the approach of integrating static and dynamic features showed promise for malware detection in general. The proposed system achieved a high accuracy rate of 99% using various types of deep neural networks (DNNs).

2.1.5) Literature Survey 6

In 2021, Darem et al. introduced a semi-supervised approach to detect obfuscated malware using deep learning (Darem et al., 2021). The proposed method used image representations of extracted features and a convolutional neural network classifier. The authors treated the problem as a semi-supervised learning task, where labels were generated for training files and predicted for testing files. The method focused on utilizing OpCode as the primary identifier of malware behavior. Experimental testing demonstrated that the proposed approach achieved an accuracy of 0.9912. Notably, the research created its own dataset specifically for this study, instead of relying on publicly available datasets from previous works.

2.1.6)Literature Survey 7

In 2021, Demetrio et al. conducted an experimental evaluation focusing on evasion techniques employed by attackers to bypass machine learning–based malware detection systems that utilize static analysis features (Demetrio et al., 2021). The study examined features extracted from PE files through static analysis, without executing the actual executables, which are typically relied upon by detection systems. The authors discussed evasion techniques employed by attackers in this context. Furthermore, the research introduced three novel attacks named Full DOS, Extend, and Shift, which involved practical manipulations that preserve the functionality of the malware. These attacks specifically targeted machine learning–based malware detection systems, aiming to evade their detection capabilities.

2.1.8) Literature Survey 8

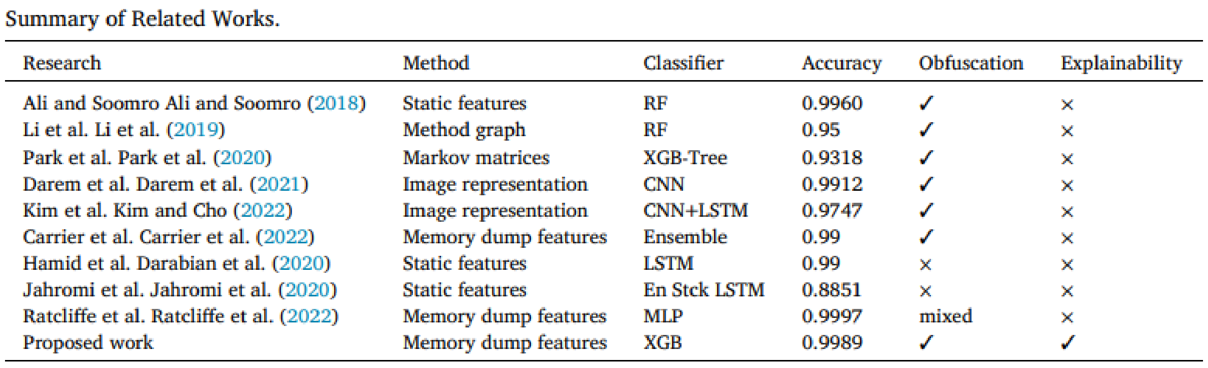
In 2022, Kim et al. introduced a hybrid deep generative model designed to detect malware variants by leveraging both global and local features (Kim and Cho, 2022). The proposed model used image representations of malware to capture global features, while local features were extracted from binary code sequences. These features were then used to train a DNN for the purpose of malware detection. The impact of the proposed method was evaluated using class activation maps. For experimentation, the Kaggle Microsoft Malware Classification Challenge dataset from 2015 was employed, indicating that it does not encompass the most recent malware families demonstrated an accuracy of 0.9747.

2.1.9) Literature Survey 9

In 2022, Carrier et al. introduced an ensemble machine learning– based system for detecting obfuscated malware (Carrier et al., 2022). The system utilized features extracted from memory dumps that contained obfuscated malware. During testing, the proposed system achieved an accuracy of 0.99 by employing an ensemble approach with naive Bayes (NB), RF, and decision tree (DT) as base learners, and logistic regression (LR) as meta learners. The publication also introduced the dataset used in our research.

2.1.10) Literature Survey 10

Ratcliffe et al. introduced another machine learning–based detection system for obfuscated malware in Ratcliffe et al. (2022). The authors tested multiple classifiers and created their own dataset using a tool named VolMemLyzer, which is the same tool used to extract data from the dataset used in our research



Summary of Related Works.

Research	Method	Classifier	Accuracy	Obfuscation	Explainability
Ali and Soomro Ali and Soomro (2018)	Static features	RF	0.9960	✓	×
Li et al. Li et al. (2019)	Method graph	RF	0.95	✓	×
Park et al. Park et al. (2020)	Markov matrices	XGB-Tree	0.9318	✓	×
Darem et al. Darem et al. (2021)	Image representation	CNN	0.9912	✓	×
Kim et al. Kim and Cho (2022)	Image representation	CNN+LSTM	0.9747	✓	×
Carrier et al. Carrier et al. (2022)	Memory dump features	Ensemble	0.99	✓	×
Hamid et al. Darabian et al. (2020)	Static features	LSTM	0.99	×	×
Jahromi et al. Jahromi et al. (2020)	Static features	En Stck LSTM	0.8851	×	×
Ratcliffe et al. Ratcliffe et al. (2022)	Memory dump features	MLP	0.9997	mixed	×
Proposed work	Memory dump features	XGB	0.9989	✓	✓

2.2) Literature Survey Observations

The Advantages of these research paper, including:

1. Advancement of Knowledge
2. Validation of Ideas
3. Peer Review
4. Publication and Recognition
5. Career Advancement
6. Contribution to the Society

CHAPTER-3
REQUIREMENT ANALYSIS AND SOLUTION APPROACH

3.1 Overall description of the project

The project focuses on developing a Memory-Based Explainable Obfuscated-Malware Detector using the Streamlit framework. The primary goal is to create a user-friendly interface that integrates machine learning algorithms to detect and explain obfuscated malware in computer memory. The project leverages various libraries, including NumPy, Pandas, Matplotlib, Scikit Learn, XGBoost, and SHAP for data manipulation, visualisation, and machine learning model interpretation.

3.2 Requirement Analysis

- User Interface Development:
    - Utilize the Streamlit framework to create an interactive and responsive user interface.
    - Ensure cross-browser compatibility for widespread accessibility.
- Data Preprocessing and Exploration:
    - Load the obfuscated-malware dataset ('Obfuscated-MalMem2022.csv') and preprocess it by dropping unnecessary columns ('Category') and encoding the 'Class' column.
    - Split the dataset into features (X) and target variable (y).
    - Use label encoding for the 'Class' column.
- Machine Learning Model Integration:
    - Implement machine learning classifiers, including Gaussian Naive Bayes, XGBoost, Random Forest, and Decision Tree.
    - Train and evaluate these classifiers using relevant metrics such as accuracy, precision, recall, and F1 score.
    - Assess the training and testing times for each algorithm.

3.3 Solution Approach

The project utilizes various machine learning algorithms and interpretable methods to detect and explain obfuscated malware. Key steps in the solution approach include:

- Data Loading and Preprocessing:
    - Load the obfuscated-malware dataset and preprocess it by dropping unnecessary columns and encoding the target variable.
- Model Training and Evaluation:
    - Train multiple classifiers (Gaussian Naive Bayes, XGBoost, Random Forest, Decision Tree) using the preprocessed dataset.
    - Evaluate each classifier's performance in terms of accuracy, precision, recall, and F1 score.
    - Record training and testing times for performance assessment.
- Feature Importance Explanation:
    - Utilize SHAP (SHapley Additive exPlanations) to explain feature importance.
    - Generate summary plots and individual feature plots to interpret the impact of selected features on model predictions.

Integrating Libraries:

NumPy Library:

NumPy is a Python library designed for scientific computing, enabling the manipulation of large, multi-dimensional arrays and matrices. It is widely used in physics, engineering, finance, and data science. Notable features include efficient array operations outperforming Python lists, broadcasting for simplified code across arrays, and comprehensive support for linear algebra operations.

NumPy has several significant features, including:

- Efficient array operations
- Broadcasting
- Linear algebra
- Fourier transforms
- Random numbers generating

Pandas' Library:

It is a Python software package for decrypting and analysing data. It manages number tables and time series using data structures and functions.

Matplotlib Library:

Many people, including John Hunter, created the matplotlib Python library for creating graphs, charts, and high-quality statistics. It is fantastic that the library can update relatively little knowledge about mathematics. Matplotlib's primary principles and actions include:

- Picture
- Structure
- Axis

XGBoost:

- Utilize XGBoost as one of the machine learning algorithms for obfuscated-malware detection due to its effectiveness in handling complex datasets.

SHAP (SHapley Additive exPlanations):

- Use SHAP for explaining feature importance and visualizing the impact of selected features on model predictions.

Scikit Learn Library:

The Python Scikit-learn library is a powerful tool for machine learning, offering various mathematical modeling methods like division, deceleration, integration, and size reduction. It provides a wide range of functionality, including supervised learning algorithms such as linear models, SVM, and decision trees. While it is not suitable for reading, tricking, or summarizing data, Scikit-learn excels in assisting with tasks like translating the spread. Its extensive capabilities make it a valuable resource for scientists working on supervised learning challenges, contributing to the widespread adoption of machine learning algorithms.

CHAPTER-4
MODELING AND IMPLEMENTATION DETAILS

4.1 Design Diagram

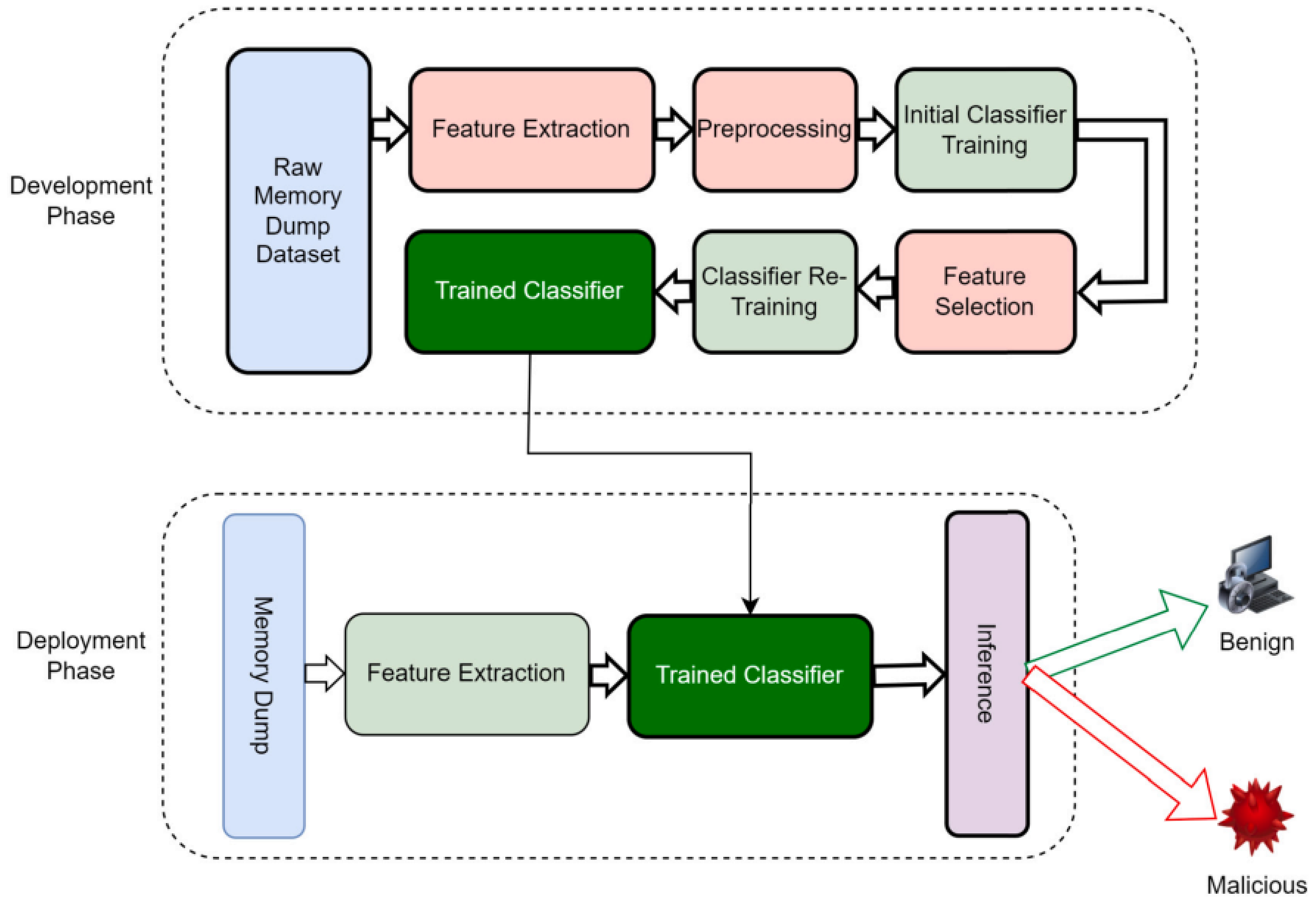


Figure 1. Overview of Proposed System.

4.2 Implementation Details

4.2.1) Dataset

The dataset used in our experiments was introduced in Carrier et al. (2022). It comprises a collection of features extracted from memory captures containing both malware and benign samples. The dataset includes 58,596 instances, with an equal distribution of 29,298 benign samples and 29,298 malware-infected samples. In the dataset, each instance corresponds to the features extracted from a single memory dump file. Each data record consists of 55 features extracted from a particular memory capture file. Additionally, an extra feature was included to provide information about the specific malware category and family, as annotated by the dataset authors. The tool used to extract these features was VolMemLyzer (Ahlashkari, 2022), which uses the Volatility Framework (The Volatility Foundation, 2022) to extract the 55 features from each memory image. These features are obtained by analyzing the memory image and capturing various information, including the number of running processes, average number of threads per process, number of open files, number of open dynamic-link libraries (DLLs), etc. For a comprehensive list of these features, please refer to The Volatility Foundation (2022); Ahlashkari (2022). As described in Carrier et al. (2022), the dataset included malicious samples that were categorized into three types: trojans, spyware, and ransomware. Fig. 2 provides an overview of the distribution of these malware categories within the dataset, displaying the respective percentages for each category. Within each of the malware categories, there were several specific malware families present in the dataset. Examples of these families include Zeus, Emotet, Gator, Transponder, Conti, MAZE, and Shade. Fig. 3 provides a breakdown of the percentage distribution for each malware family within the dataset. The figures indicate that the different categories and families are adequately represented, without significant discrepancies that could introduce bias into the detection model towards any particular malware family. The analyzed malware families in the dataset exhibited varying levels of obfuscation techniques.

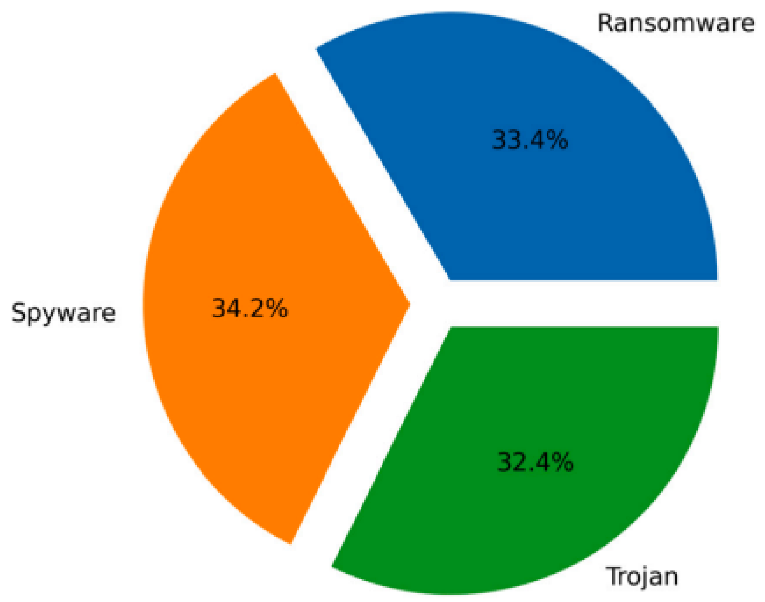


Figure 2. Distribution of Malware Categories in the MalMem Dataset.

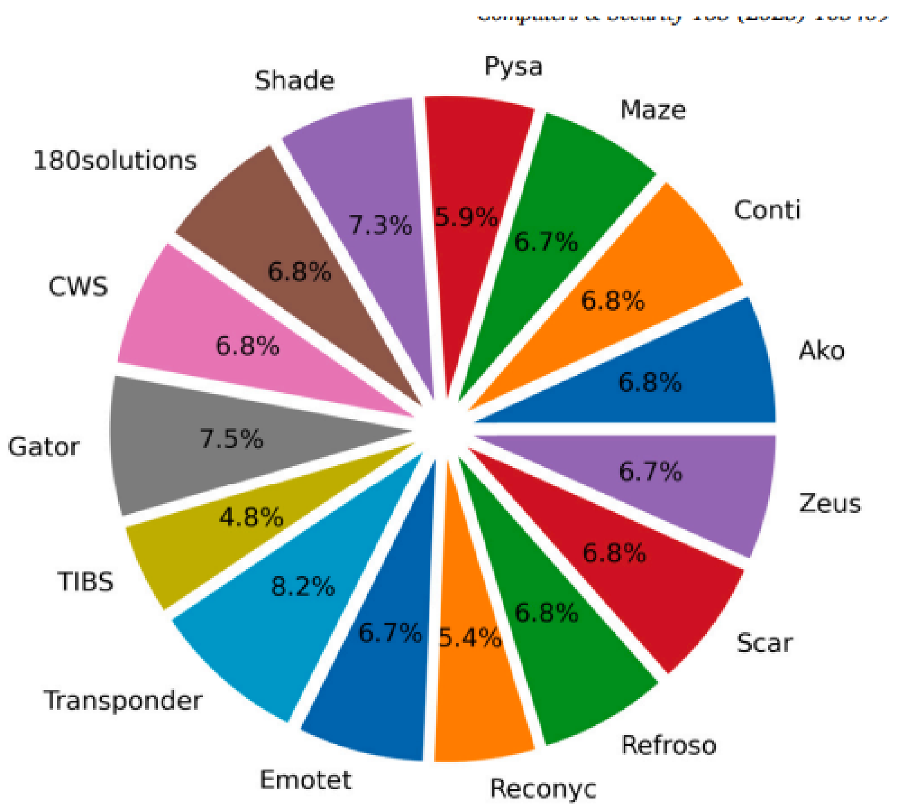


Figure 3. Distribution of Malware Families in MalMem Dataset

4.2.2) Data Pre-processing

After conducting a thorough analysis of the dataset, we discovered that it exhibited a balanced distribution. Fig. 2 demonstrated that the dataset contained a relatively equal representation of the three types of malware, with no significant discrepancies in their proportions. Similarly, Fig. 3 revealed a balanced distribution of various malware families, ensuring that no specific family was over-represented compared to others. Furthermore, our examination confirmed that the dataset was complete, containing no missing data. This indicated that the dataset was well-prepared for the training phase and did not require any additional preprocessing steps.

4.2.3) Model Building

Model building is the process of developing a mathematical model that will aid in predicting or estimating future events based on data obtained in the past.

Steps in Model Building

The following steps are involved:

● Algorithm Selection

● Training Model

● Prediction / Scoring

Algorithm Selection

Algorithms:

1.Recursive Feature Elimination Using Feature Importance

2.10-Fold Cross-Validation

3. Decision Tree

4. Gaussian Naive Bayes (GNB)

5. Extreme gradient boost (XGB).

Training Model

In machine learning, training a model entails using an algorithm to understand the connection between inputs (features) and outputs (labels) in a given dataset. The purpose of training is to build a model that can generalize to new, previously unknown data and generate correct predictions.

Several steps are included in the training process, including:

1. Data Preparation: This includes cleaning, pre-processing, and structuring the data so that it may be utilized for training.
2. Feature Selection: This is the process of deciding which features will be utilized to train the model.
3. Model selection: It entails picking the best method or model architecture for the particular situation at hand.
4. Training: It entails feeding the model training data and changing the model's parameters to minimize the error or loss function.
5. Validation: This is about putting the trained model through its paces on a different validation dataset to check that it generalizes effectively to new data.
6. Hyper-parameter Tuning: Tuning the model's hyperparameters to optimize its performance on the validation dataset is known as hyperparameter tuning.

7. Testing: Finally, the model is evaluated on a different test dataset to assess its performance on fresh, previously unknown data. Therefore, to construct a model that can reliably predict the output for fresh inputs, the process of training a model in machine learning comprises a mix of data preparation, feature selection, model selection, and parameter tweaking.

Therefore, to construct a model that can reliably predict the output for fresh inputs, the process of training a model in machine learning comprises a mix of data preparation, feature selection, model selection, and parameter tweaking.

4.2.4) Methods of Machine Learning

Obligations are frequently separated into big groups in machine learning. These categories are entirely dependent on how data is obtained and how the system responds to it.

Unsupervised learning, which provides the algorithm with no labeled data so that it can discover structure in its input statistics, and supervised learning, which trains algorithms primarily based on example input and output data that is labeled by humans, are two of the most widely used machine learning methods. Let's dig further into each of those tactics.

★ Supervised Machine Learning

This machine learning model is given a dataset with inputs and their right outputs. Alternatively, labeled datasets are fed into machine learning algorithms for training (directed training).

Applications of supervised learning include speech recognition, spam detection, bioinformatics, and other fields.

A kind of machine learning in which the algorithm is taught with a labeled dataset is known as supervised machine learning. The labeled dataset contains inputs (features) and their associated outputs (labels), and the algorithm's purpose is to learn the relationship between the inputs and the outputs in order to predict the output for fresh inputs.

During training, the algorithm is given the right answer (labeled data) and attempts to learn a mapping from the inputs to the outputs that can generalize to new, unseen inputs. This sort of learning is often employed in image recognition, audio recognition, and natural language processing applications.

For supervised machine learning, many techniques such as decision trees, logistic regression, support vector machines, and neural networks can be employed. The algorithm used is determined by the specific challenge at hand as well as the features of the dataset.

Overall, supervised machine learning has been widely applied in a variety of sectors and has demonstrated significant effectiveness in tackling a wide range of real-world issues.

★ Unsupervised Machine Learning

Labeled datasets are not supplied in this learning. It attempts to detect patterns in the data in the datasets. Human or human supervision is required less in this sort of learning than in supervised learning.

It can more readily organize and handle unstructured and unlabeled data. However, it makes analyzing and discovering patterns in complicated data simpler.

Unsupervised machine learning is a sort of machine learning in which the algorithm is trained on a dataset that has not been labeled. In contrast to supervised learning, no labels are supplied for the data, so the algorithm must identify patterns and structure on its own.

Unsupervised learning involves providing the algorithm with a collection of inputs (features) and attempting to detect patterns or similarities between the inputs without any prior knowledge of the output or label. This form of learning is often employed in clustering, anomaly detection, and dimensionality reduction applications.

Unsupervised learning involves providing the algorithm with a collection of inputs (features) and attempting to detect patterns or similarities between the inputs without any prior knowledge of the output or label. This form of learning is often employed in clustering, anomaly detection, and dimensionality reduction applications.

4.2.5) Algorithms of Machine Learning Used

Recursive Feature Elimination Using Feature Importance

Recursive Feature Elimination (RFE) is a feature selection technique employed in the context of a Memory-Based Explainable Obfuscated-Malware Detector. Unlike Random Forest's application in agriculture, RFE focuses on selecting the most informative features from memory-based data to enhance the interpretability and efficiency of the malware detection model.

Role in the Project:
RFE is instrumental in streamlining the model by recursively eliminating less important features, thereby improving the model's efficiency and interpretability. In the domain of malware detection, where memory-based features play a crucial role, RFE helps identify the key characteristics associated with malicious behavior.
Application to Malware Detection:

When applied to the detection of obfuscated malware in memory snapshots, RFE aids in selecting the most relevant features. These features might include specific memory patterns, API calls, or execution behaviors indicative of malicious activity. By iteratively removing less significant features, RFE contributes to creating a focused and interpretable model.

Benefits:

Efficiency: RFE streamlines the model, potentially reducing computational resources and improving real-time detection capabilities.

Interpretability: By selecting a subset of the most relevant features, RFE facilitates a clearer understanding of the memory-based characteristics contributing to the model's decisions.

10-Fold Cross-Validation

Description:
10-Fold Cross-Validation is a model evaluation technique applied to the Memory-Based Explainable Obfuscated-Malware Detector. This method involves dividing the dataset into ten subsets or folds, training the model on nine of them, and testing its performance on the remaining fold. This process is repeated ten times, with each fold serving as the test set exactly once.

Role in the Project:
The primary role of 10-Fold Cross-Validation is to assess the performance of the Memory-Based Explainable Obfuscated-Malware Detector across multiple subsets of the data. This helps in obtaining a more robust estimate of the model's generalization capabilities, ensuring that the detector performs well on diverse data samples.
Application to Malware Detection:

In the context of malware detection, 10-Fold Cross-Validation allows the model to be trained and tested on various combinations of memory-based features, enhancing its ability to generalize to different patterns of obfuscated malware. It helps identify whether the model is consistently effective across diverse subsets of the data.

Benefits:

Robust Evaluation: Cross-validation provides a more robust evaluation of the model's performance by testing it on different subsets of the data.

Reduced Bias: By using multiple test sets, 10-Fold Cross-Validation helps reduce bias in performance metrics, providing a more representative estimate.

Decision Tree

Description:
Decision Tree is a fundamental machine learning algorithm used in the Memory-Based Explainable Obfuscated-Malware Detector. It creates a tree-like structure where each node represents a decision based on a feature, ultimately leading to a prediction. In malware detection, a Decision Tree may analyse memory-based features to classify instances as benign or malicious.

Role in the Project:
Decision Tree is employed to model the decision-making process of the Memory-Based Explainable Obfuscated-Malware Detector. It is particularly useful for understanding how specific memory-based characteristics contribute to the identification of obfuscated malware.
Application to Malware Detection:

The Decision Tree algorithm, when applied to memory snapshots, can analyze various features such as memory patterns, system calls, or code execution behaviors. By visually inspecting the resulting tree, security analysts can gain insights into the decision logic, aiding in the interpretability of the model.

Benefits:

Interpretability: Decision Trees provide a transparent decision-making process, making it easier to interpret how specific features contribute to the final decision.

Visualisation: The tree structure allows for visual representation, aiding in the communication of the model's logic to stakeholders.

Gaussian Naive Bayes (GNB)

Description:
Gaussian Naive Bayes (GNB) is a probabilistic machine learning algorithm applied in the context of the Memory-Based Explainable Obfuscated-Malware Detector. It is based on Bayes' theorem and assumes that features are conditionally independent, given the class label. GNB can model the probability distribution of memory-based features to classify instances.

Role in the Project:
GNB is utilized to model the probabilistic relationships between memory-based features and the likelihood of malware presence. It calculates the probability of an instance belonging to a particular class, aiding in classification.
Application to Malware Detection:
In malware detection, GNB can assess the likelihood of specific memory patterns or behaviors being associated with malicious activity. It provides a probabilistic framework for evaluating the presence of obfuscated malware based on memory-related characteristics.
Benefits:
Efficiency: GNB is computationally efficient and can perform well with relatively small datasets.
Probabilistic Inference: The probabilistic nature of GNB allows for a nuanced assessment of the likelihood of malware presence.

Extreme gradient boost (XGB)

Description:

Extreme Gradient Boost (XGB) is a powerful gradient boosting algorithm utilized in the Memory-Based Explainable Obfuscated-Malware Detector. It belongs to the ensemble learning family and sequentially builds a series of weak learners, optimizing their combination to enhance predictive performance.

Role in the Project:

XGB is employed to boost the performance of the Memory-Based Explainable Obfuscated-Malware Detector. It iteratively corrects errors made by preceding models, focusing on improving the detection accuracy of obfuscated malware based on memory features.

Application to Malware Detection:

XGB excels in identifying complex patterns and relationships within memory-based data. In the context of malware detection, it can capture subtle nuances in memory snapshots that may indicate obfuscated malware, contributing to a more accurate and robust detection model.

Benefits:

High Predictive Accuracy: XGB often achieves high predictive accuracy, especially in scenarios where intricate relationships exist between features.

Robust to Overfitting: XGB is designed to handle overfitting, a common challenge in complex datasets.

Random Forest

Description:

Random Forest is a robust machine learning algorithm employed in the Memory-Based Explainable Obfuscated-Malware Detector. It falls under the category of ensemble learning, where multiple decision trees are combined to improve overall predictive performance and generalization.

Role in the Project:

Random Forest plays a crucial role in enhancing the predictive accuracy and reliability of the Memory-Based Explainable Obfuscated-Malware Detector. By aggregating predictions from a multitude of decision trees, it provides a more robust and stable model for detecting obfuscated malware based on memory-based features.

Application to Malware Detection:

In the context of malware detection, Random Forest excels in handling large and complex datasets that characterize memory snapshots. It can effectively analyze various memory-related features, such as patterns, system calls, or code execution behaviors, contributing to the identification of obfuscated malware.

Benefits:

Ensemble Learning: Random Forest combines the strength of multiple decision trees, mitigating the risk of overfitting and improving generalization.

Feature Importance: The algorithm provides insights into the importance of different memory-based features, aiding in the interpretability of the detection model.

4.3 Risk Analysis and Mitigation

Data Quality and Availability:

- **Risk:** Incomplete or inaccurate data can compromise the effectiveness of the malware detection model, leading to potential false positives or negatives.
- **Mitigation:** Collaborate with cybersecurity experts and domain specialists to ensure the quality, relevance, and accuracy of the data. Implement rigorous data preprocessing techniques to address any inconsistencies or missing values in the memory-based features.

Algorithm Selection:

- **Risk:** Choosing an inappropriate algorithm for obfuscated-malware detection may result in suboptimal performance and reduced model interpretability.

- **Mitigation:** Conduct thorough benchmarking and comparative analysis of different algorithms, considering factors such as accuracy, interpretability, and computational efficiency. Run pilot studies on a smaller scale to assess the performance of various algorithms before making a final selection for full-scale implementation.

Model Overfitting:

- **Risk:** Overfitting occurs when the model performs well on the training data but fails to generalize to new, unseen data, impacting the detector's reliability.
- **Mitigation:** Implement robust techniques like 10-fold cross-validation and regularization during the model training phase. Regularly monitor and evaluate the model's performance on a separate validation dataset to identify and address potential overfitting issues.

Dependency on Memory Data:

- **Risk:** The malware detector's performance is highly dependent on memory-based features, and changes in malware tactics may affect the model's accuracy.
- **Mitigation:** Stay informed about emerging obfuscation techniques and malware trends. Continuously update the model by incorporating new memory-based features and patterns. Collaborate with cybersecurity communities to enhance the model's adaptability to evolving malware threats.

Interpretability and Explainability:

- **Risk:** Lack of interpretability in the model's decision-making process may hinder its acceptance and trust among cybersecurity professionals.
- **Mitigation:** Utilize explainable AI techniques, such as SHAP values and feature importance plots, to provide insights into the model's decision logic. Ensure that the model's predictions can be easily understood and interpreted by cybersecurity analysts.

Chapter - 5
TESTING

5.1) Testing Plan

The experiments conducted in this research can be divided into the following steps:

1. Initial training and testing: The dataset is initially split randomly, with proper stratification, into a 66.66% training subset and a 33.33% testing subset. A pipeline of five classifiers is trained and tested to determine the best-performing classifier among them.

2. Feature selection: The goal of this step is to select the minimum number of effective features in order to improve system efficiency and create a lightweight system. The RFE algorithm based on feature importance is used for feature selection. Algorithm 1 outlines the steps of the RFE algorithm, which involves iteratively eliminating the feature with the least importance, retraining the model, and recalculating feature importance until a desired number of features is reached while maintaining a minimum compromise in accuracy (Guyon et al., 2002).

3. Post-selection training and testing: The dataset with the selected features is used to retrain and retest the five classifiers, assessing the impact of the feature selection process on the performance metrics.

4. Validation: The model undergoes 10-fold cross-validation in this step. Algorithm 2 describes the steps involved in 10-fold crossvalidation. The dataset is randomly divided into 10 folds, with each fold used for testing once while the remaining 9 folds are used for training. If the performance metrics obtained from the 10 folds are consistent and exhibit a small standard deviation, it indicates that the classifier is generalizing well and is not suffering from overfitting.
5. Model's explainability: In this step, the model undergoes an explainability analysis to ensure that the model's predictions are not based on black-box operations. Further details about the explainability technique employed are provided in Section

5.2) Initial results

As described in Section 5.1, the first stage of the experimentation involved the creation and evaluation of five different classifiers to identify the best-performing one. The chosen classifier algorithms chosen for the initial experiments were:

1. RF
2. DT
3. Gaussian Naive Bayes (GNB)
4. Extreme gradient boost (XGB)

These four algorithms were chosen over deep learning methods due to the computationally intensive nature of deep learning compared to the listed algorithms.

One of the main objectives of the proposed system is to create a lightweight system. Therefore, opting for less resource-intensive classification algorithms aligns with our design goals and facilitates their achievement. The 55-feature dataset was randomly divided into a training subset comprising 66.66% of the data and a testing subset comprising 33.33% of the data, with stratification, to maintain the balance between malicious and benign samples. The performance metrics obtained from the testing phase are presented in Table 3. As shown in the table, all five classifiers demonstrated high accuracy scores, surpassing 0.99. Among them, the RF classifier achieved slightly higher accuracy. However, when considering both accuracy and timing, the XGB classifier exhibited superior performance.

5.3) Post-selection results

The resulting 5-feature dataset obtained after feature selection was further divided into a training subset (66.66%) and a testing subset (33.33%). The training subset was utilized to retrain the same set of five classifiers. The performance metrics obtained from the subsequent tests conducted after training are presented in Table 3

The table demonstrates that the performance of each classifier remained consistently high even with the reduction in the number of features from 55 to only 5. This indicates that the selected five features hold significant importance in the classification process. Notably, when considering both accuracy and timing parameters, XGB outperformed the other classifiers slightly in terms of timing, while maintaining a similar level of accuracy. The testing time of the XGB classifier exhibited a significant improvement, as anticipated, after feature selection. The testing time for a single instance decreased from 0.569 μs with 55 features to 0.413 μs with the 5-feature predictions. This represents a 26% improvement in timing, further highlighting the practical advantage of the feature selection process.

.

CHAPTER-6

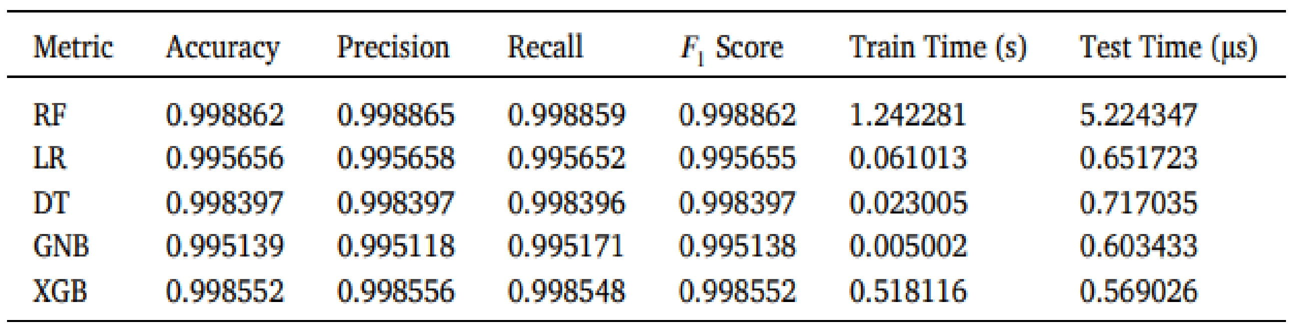
FINDINGS,CONCLUSION AND FUTURE WORK

7.1) Findings

Designing an explainable obfuscated malware detector involves creating a system that not only identifies memory-based malware but also provides clear explanations for its decisions. Explainability is crucial for building trust in the detection system and for facilitating analysis by security experts. Here are some key findings and considerations for a Memory-Based Explainable Obfuscated Malware Detector:

Table1

Performance Metrics for 55-Feature Dataset.



Metric	Accuracy	Precision	Recall	F_1 Score	Train Time (s)	Test Time (µs)
RF	0.998862	0.998865	0.998859	0.998862	1.242281	5.224347
LR	0.995656	0.995658	0.995652	0.995655	0.061013	0.651723
DT	0.998397	0.998397	0.998396	0.998397	0.023005	0.717035
GNB	0.995139	0.995118	0.995171	0.995138	0.005002	0.603433
XGB	0.998552	0.998556	0.998548	0.998552	0.518116	0.569026

Feature selection

We decided to use RFE for feature selection instead of other statistical dimensionality reduction algorithms, such as principal component analysis (PCA), singular value decomposition (SVD), and linear discriminant analysis (LDA) for several reasons. Unlike dimensionality reduction algorithms such as PCA, SVD, and LDA, RFE has certain advantages in the context of feature selection. While the mentioned dimensionality reduction algorithms reduce the number of features used as input to the classifier, they still require capturing the same number of features at the data acquisition stage in real-life deployments. Additionally, these algorithms involve mathematically intensive operations during preprocessing to generate the new features at deployment. On the other hand, RFE not only reduces the number of features fed into the classifier but also reduces the number of features captured during data acquisition without the need for additional preprocessing. This results in improved efficiency and reduced processing time.

Table 2

Selected Features.

Feature	Description
`svcscan.nservices`	Total number of services registered in a memory image
`dlllist.avg_dlls_per_proc`	Average number of DLLs called per process
`handles.nmutant`	Total number of mutant handles detected
`svcscan.kernel_drivers`	Total number of processes running in kernel mode that have an associated driver name
`svcscan.shared_process_services`	Total number of services that share one or more process with other services

Table 3

Performance Metrics for 5-Feature Dataset.

Metric	Accuracy	Precision	Recall	F_1 Score	Train Time (s)	Test Time (μs)
RF	0.998914	0.998917	0.998910	0.998914	1.268830	5.224335
LR	0.995656	0.995658	0.995652	0.995655	0.061013	0.516630
DT	0.998397	0.998397	0.998396	0.998397	0.027005	0.551735
GNB	0.995139	0.995118	0.995171	0.995138	0.006000	0.510723
XGB	0.998552	0.998556	0.998548	0.998552	0.477106	0.413808

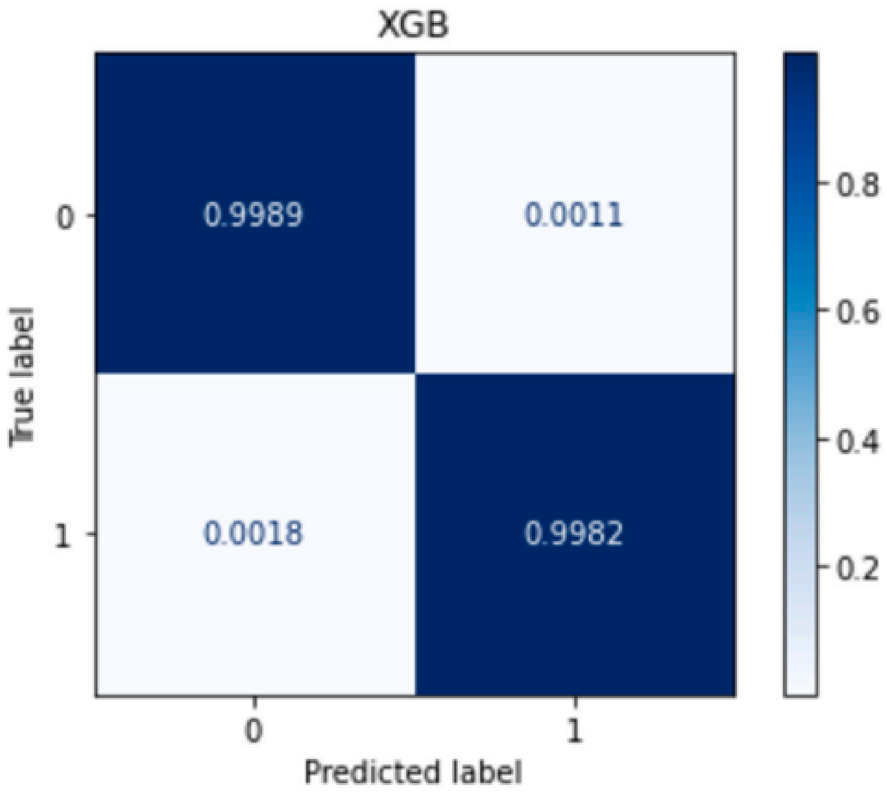
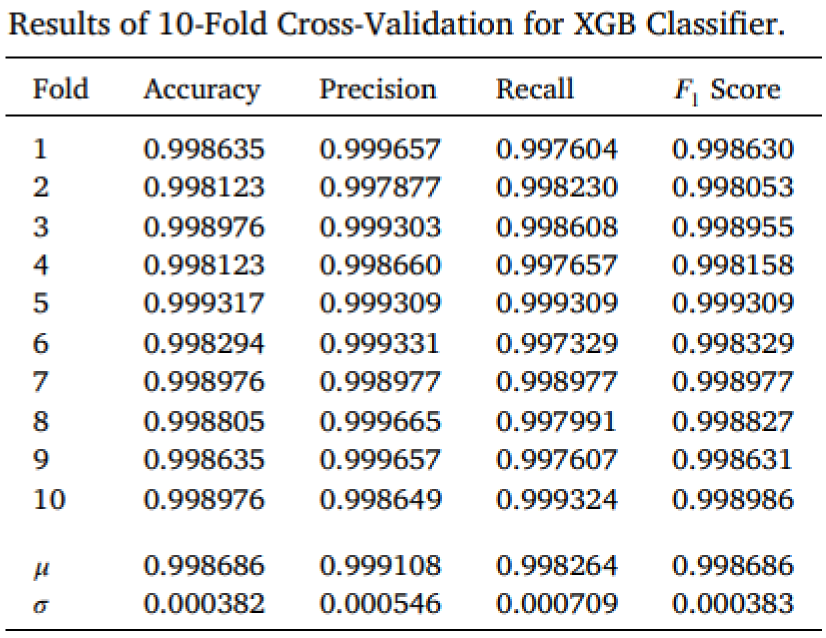


Fig. 4. Confusion Matrix Plot for XGB Classifier.

Cross-validation

As a final step to validate the proposed model, we performed 10-fold cross-validation Table 6 shows the results of this process. As shown in the table, the mean value of the performance metrics obtained from the 10-fold cross-validation aligns closely with the earlier results presented. Moreover, the low standard deviation across the performance metrics signifies the robustness and reliability of the proposed model. These findings indicate that the model's performance extends beyond the specific training dataset, suggesting a strong generalization capability. To prevent information leakage and ensure the integrity of the crossvalidation process in the MalMem dataset, a specific strategy was employed to handle the potential issue of multiple memory dumps from the same malware family. The strategy involved ensuring that the 10-fold cross-validation splits were carefully designed to avoid any intersection of malware families within each fold. This precautionary step was implemented before initiating the first cycle of cross-validation, thereby mitigating any possibility of information leakage and maintaining the integrity of the evaluation process

Table 4



Results of 10-Fold Cross-Validation for XGB Classifier.

Fold	Accuracy	Precision	Recall	F_1 Score
1	0.998635	0.999657	0.997604	0.998630
2	0.998123	0.997877	0.998230	0.998053
3	0.998976	0.999303	0.998608	0.998955
4	0.998123	0.998660	0.997657	0.998158
5	0.999317	0.999309	0.999309	0.999309
6	0.998294	0.999331	0.997329	0.998329
7	0.998976	0.998977	0.998977	0.998977
8	0.998805	0.999665	0.997991	0.998827
9	0.998635	0.999657	0.997607	0.998631
10	0.998976	0.998649	0.999324	0.998986
μ	0.998686	0.999108	0.998264	0.998686
σ	0.000382	0.000546	0.000709	0.000383

Model's explainability

Our research aimed to explain the factors that influence the prediction decisions made by the trained model. The goal was to enhance trust in the model by providing transparency in its operation. By gaining insights into the inner workings of the model and uncovering the conditions and factors that contribute to its high accuracy, we aimed to ensure that the model's performance is based on explainable and interpretable processes, rather than relying on opaque"black-box" operations. To explain our proposed model, we utilize Shapley additive explanation (SHAP). SHAP is a model-agnostic method introduced in 2017 that aims to explain machine-learning models. SHAP provides explanations derived from game theory. The SHAP process involves quantifying the impact of each feature by comparing the model's performance with and without the presence of that feature in the data. This approach allows us to gain insights into the influence of individual features on the model's prediction outcomes.

In our experiment, we employed the TreeExplainer as the explainer type. Fig. 5 presents a summary plot of SHAP values for the five selected features, arranged in descending order based on their impact on the prediction decisions. The feature with the highest influence is positioned at the top, while the feature with the lowest influence is at the bottom. The values displayed on the left side of the axis have a downward drag on the prediction value, resulting in it being closer to the “benign” category. Conversely, the values on the right side of the axis exert an upward push on the prediction value, bringing it closer to the “malware” category. In the plot, red dots correspond to high feature values, while blue dots correspond to low feature values. In the case of binary features, red dots represent a value of 1, while blue dots represent a value of 0. As depicted in Fig. 5, the feature svcscan.nservices has the most significant impact on the prediction. This feature represents the total number of services detected in the memory image. The presence of red dots on the left side of the plot indicates that a high number of services is associated with a lower prediction value, indicating a higher likelihood of being classified as “benign.” On the other hand, the presence of blue dots indicates that a lower value of the registered service feature leads to an increase in the prediction value towards “malware.” This observation is consistent with the behavior patterns observed in recent malware. Instead of running as independent services, modern obfuscated-malware families often employ techniques such as process hijacking or fileless execution . Process hijacking involves malicious code tricking other processes into running them or taking control of them. As a result, the number of processes running independently may be lower in the presence of malware. One common target for process hijacking in the Windows operating system is “svchost.exe”. Additionally, obfuscatedmalware code may be “fileless,” meaning it is stored within registry entries or executed through scripting languages like PowerShell, rather than being present as a standalone file. This technique aims to evade detection by traditional file-based scanning approaches (Cybertriage, 2021). The next two features that have a significant impact on the prediction are called dlllist_avg_dlls_per_proc and handles.nmutant. These features have a similar effect, where higher values bring the prediction closer to being classified as benign, while lower values indicate a higher likelihood of being classified as “malware.” Similar to svcscan.nservices, lower values of these two features are associated with a malware infection. When a computer is infected with malware, the average number of DLL files invoked by a process tends to decrease. This is because the malware hijacks normal processes, preventing them from functioning properly. However, the obfuscated malware itself may still call other DLLs, particularly those used in modifying the Windows registry. Nevertheless, on average, the malware uses fewer DLLs compared to a benign process. The same behavior of the malware that leads to lower values of handles.nmutant is observed when a computer is infected. The fourth feature depicted in the figure is referred to as svcscan.kernel_drivers. The figure illustrates that there is a specific range of high values for this feature that tends to associate the prediction more closely with malware. Additionally, there is a distinct range of low values on the right side of the figure that also contributes to the prediction leaning towards malware. This suggests that this particular feature alone is not highly conclusive in supporting the decision and is more effective when considered in combination with other features. The reason for the feature's limited standalone effectiveness is that many newly developed obfuscated-malware families adopt a strategy of running their processes in kernel mode using a driver name, rather than operating in user mode with a recognizable EXE identifier. This approach allows malware to evade detection since most malware detection software focuses on identifying user mode processes with distinct EXE identifiers. Consequently, a slightly higher number of processes running in kernel mode would logically indicate a potential malware infection.

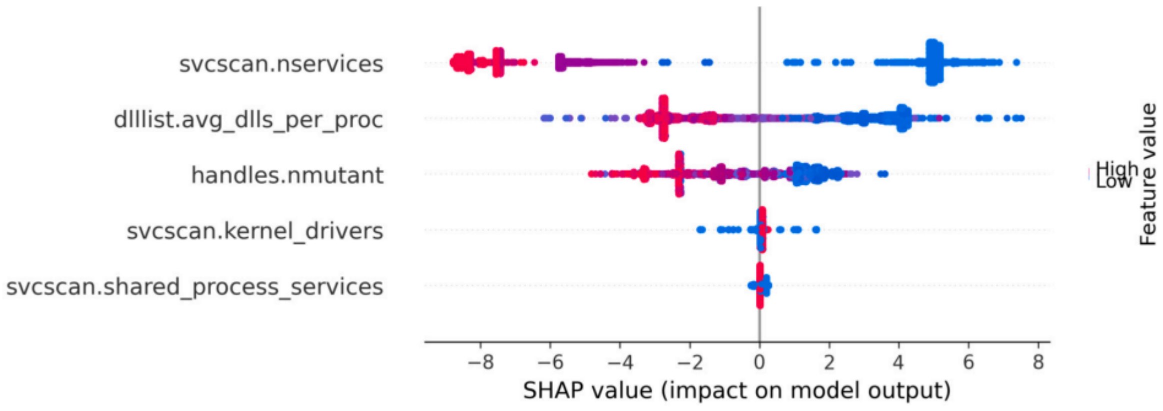


Figure 5. SHAP Values Plot for Selected Features.

6.2)Conclusion:

This paper introduced a lightweight obfuscated-malware detector based on explainable machine-learning techniques The system was evaluated using the MalMem2022 dataset and achieved a remarkable accuracy of over 99.8% by utilizing only five carefully selected features. The feature selection method, RFE, effectively reduced the number of features while maintaining high accuracy. This streamlined approach significantly enhanced the efficiency of the system. Additionally, the detection time of the proposed system was impressively fast, measuring 0.413 µs. To ensure transparency and interpretability, the proposed system was explained using SHAP values. The obtained explanations were found to be consistent with modern malware behavior. The utilization of SHAP values not only justified the high accuracy achieved by the classifier but also ensured that the system's performance was based on explainable reasoning rather than operating as a black-box approach.

6.3) Future Work:

The future work of a Memory-Based Explainable Obfuscated Malware Detector involves ongoing research, development, and enhancement to keep up with the evolving landscape of cybersecurity threats. Here are some potential directions for future work in this domain:

Feature ExtractionEnhanced :

- Investigate and incorporate new features that capture subtle characteristics of obfuscated malware.
- Explore techniques for automatically extracting relevant features from memory dumps.

Deep Learning Interpretability:

- Develop methods to improve the interpretability of deep learning models used in malware detection.
- Research techniques that provide insights into the decision-making process of complex neural networks.
- An interesting future direction would involve exploring updated datasets that include both obfuscated and non-obfuscated samples. Another potential avenue for future research involves investigating the deployment of the system on low-resource devices and addressing the task of categorizing malware into specific families to facilitate forensic analysis

References :

- Ahlashkari, 2022. VolMemLyzer:https://github.com/ahlashkari/VolMemLyzer. (Accessed 14 June 2022).
- Dataset:https://www.unb.ca/cic/datasets/malmem-2022.html
- Matplotlib — Visualization with Python, 2022. https://matplotlib.org. (Accessed 30 March 2023).
- Scikit-learn: machine learning in Python — scikit-learn 1.0.2 documentation, 2022. https://scikit-learn.org/stable. (Accessed 30 March 2022)
- https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
- https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html
- https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
- https://www.geeksforgeeks.org/cross-validation-machine-learning/
- XGBoost - GeeksforGeeks
- Welcome to the SHAP documentation — SHAP latest documentation
- shap.TreeExplainer — SHAP latest documentation (shap-lrjball.readthedocs.io)