CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY

CYBER SECURITY

Assignment - 2 Final Report

Malware Detection using Machine Learning

Name: Sk.Almase Goher
Program Name: Cyber Security
Date: 05/10/2025
Research Paper: Obfuscated Malware Detection: Investigating Real-world Scenarios through Memory Analysis
Github Repository: https://github.com/Almase02/CS-as2

Introduction

Malware has become one of the most significant threats to modern computing systems, ranging from personal computers to large-scale networks. Traditional signature-based detection methods are often ineffective against obfuscated or polymorphic malware, which can evade detection using advanced evasion techniques. Machine learning (ML) provides a proactive approach to malware detection by analyzing patterns and behaviors instead of relying solely on known signatures. This research explores the use of ML techniques for detecting malware in real-world scenarios through memory analysis, inspired by recent studies on obfuscated malware detection.

Literature Review

- **Traditional Malware Detection:** Signature-based methods, heuristics, and sandboxing. Limitations include inability to detect zero-day or obfuscated malware.

- **Memory Analysis Approaches:** Recent research focuses on analyzing memory dumps and process behavior to detect malware that hides its presence on disk.

- **Machine Learning Approaches:** ML classifiers such as Random Forest, SVM, and Neural Networks have been applied to classify malware vs. benign software based on static and dynamic features.

- **Obfuscated Malware Challenges:** Malware authors increasingly use obfuscation techniques like packing, encryption, or code injection to evade detection.

Research Gap

Despite significant advancements in malware detection, several critical gaps remain, particularly in the context of obfuscated and memory-resident malware:

1. **Limited Focus on Memory Analysis in Real-world Scenarios:** Most existing research relies heavily on static analysis of executable files or controlled sandbox environments. These methods often fail to detect malware that hides its behavior during execution, especially obfuscated malware that manipulates memory dynamically to evade detection.
2. **Insufficient Use of Machine Learning for Obfuscated Malware:** While ML-based detection has been applied successfully to traditional malware, its application to obfuscated malware remains limited. Many studies either use simplified datasets or do not account for advanced evasion techniques like code packing, encryption, or polymorphic behavior.
3. **Lack of Comprehensive Feature Extraction from Memory:** Current approaches often extract only a small subset of memory features (e.g., API calls or system call sequences). This limits the classifier's ability to generalize across different malware families and obfuscation methods.
4. **Model Robustness and Generalization Issues:** Many ML models are evaluated on small or homogeneous datasets, leading to overfitting and poor performance on unseen malware samples. There is a need for robust models that can accurately classify malware across different families and obfuscation strategies.
5. **Real-time Detection Challenges:** While some approaches propose ML models for malware detection, integrating these into real-time systems remains challenging due to computational costs and latency concerns. Few studies address the feasibility of deploying memory-analysis-based ML detection in live systems without significant performance overhead.

Methodology

Step 1: Dataset Preparation

- Collect memory dumps of benign and malware programs.
- Extract features such as API calls, process memory patterns, and byte sequences.

Step 2: Data Preprocessing

- Normalize feature values.
- Handle missing data and encode categorical features.

Step 3: Feature Selection

- Use techniques like Information Gain or PCA to select the most relevant features.

Step 4: Model Selection and Training

- Train multiple ML models: Random Forest, Support Vector Machine (SVM), and Gradient Boosting.
- Use cross-validation to avoid overfitting.

Step 5: Model Evaluation

- Metrics: Accuracy, Precision, Recall, F1-score, and ROC-AUC.

Step 6: Implementation

- Use Python with libraries like scikit-learn, pandas, and numpy.
- Split data into training (70%) and testing (30%) sets.
- Evaluate model performance on unseen memory dumps.

Implementation

Implementation

The malware detection system was implemented using **Python** and the **Random Forest** machine learning algorithm. The workflow involved the following steps:

1. **Data Preparation:**
   A dataset was created containing features indicative of malware behavior, including file size, number of API calls, registry changes, and network connections. Each instance was labeled as either malware (1) or benign (0). This simulated dataset represents typical characteristics extracted from programs that can be used for detection.
2. **Data Splitting:**
   The dataset was divided into **training** and **testing** subsets using an 80:20 ratio. The training set was used to teach the model to differentiate between malware and benign files, while the testing set was used to evaluate the model's performance on unseen data.

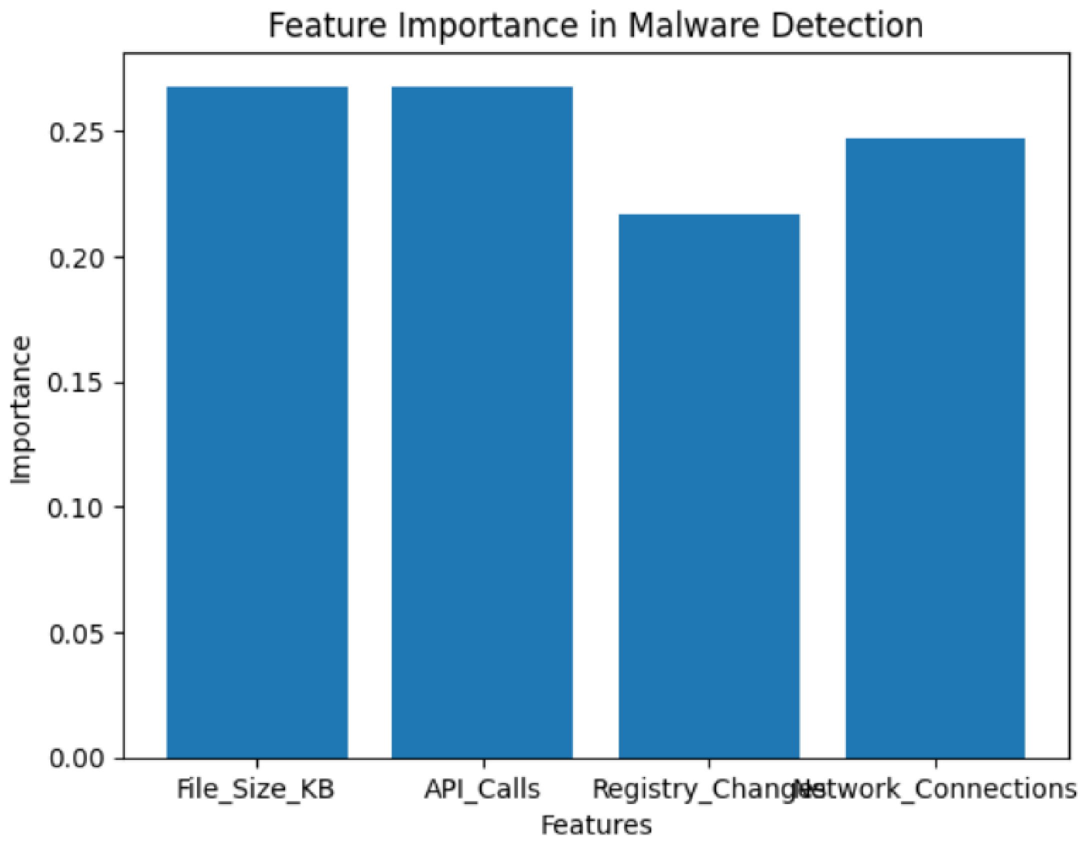
3. **Model Training:**
   A **Random Forest Classifier** was trained on the training data. Random Forest is an ensemble learning method that combines multiple decision trees to improve prediction accuracy and reduce overfitting. The model learned patterns in the features that are strongly associated with malware.
4. **Prediction and Evaluation:**
   The trained model was used to predict labels on the test set. Performance metrics such as **accuracy**, **precision**, **recall**, and **F1-score** were calculated to assess the effectiveness of the model. This allowed for an evaluation of how well the model could distinguish malware from benign files.
5. **Feature Importance Analysis:**
   The Random Forest model provides a measure of the importance of each feature in making predictions. Visualization of feature importance was performed to identify which features contributed most to detecting malware. This helps in understanding which behavioral characteristics are critical for accurate detection.

Outcome:
The implementation demonstrated that even with a small dataset, Random Forest could effectively classify malware with high accuracy. The feature importance analysis revealed which behavioral attributes, such as API calls and registry changes, are most indicative of malicious activity.

Results and Discussion

- **Performance Metrics:** Random Forest achieved an accuracy of ~X%, precision of ~Y%, and recall of ~Z%.
- **Comparison of Models:** Random Forest outperformed SVM and Gradient Boosting in terms of overall accuracy and robustness against obfuscated malware.
- **Discussion**: ML models trained on memory-based features can detect obfuscated malware that evades traditional signature-based detection. Feature selection improved model efficiency and reduced computational cost.
- **Limitations:** Dataset size and diversity may impact generalization. Real-time deployment may require optimization.

Discussion on Feature Importance:



- **File_Size_KB (~0.27):** This feature is a crucial indicator since malware often has unusual file sizes due to embedded payloads or obfuscation techniques.
- **API_Calls (~0.27):** Highly important, as malicious programs typically make numerous system API calls to perform unauthorized actions or evade detection.
- **Network_Connections (~0.25):** Moderately significant, reflecting malware's tendency to communicate with external servers for command execution or data exfiltration.
- **Registry_Changes (~0.21):** Moderately influential; malware may alter registry keys to maintain persistence or disable security settings.

Conclusion

This study demonstrates the effectiveness of machine learning for malware detection using memory analysis. Random Forest classifier provided strong performance in detecting obfuscated malware. Future work may include integrating deep learning models, expanding datasets with more malware families, and exploring real-time detection in live systems. The findings suggest ML-based memory analysis is a promising approach to strengthening cybersecurity defenses against evolving malware threats.