Individual Project Submission 2024 (Preliminary Report)

Name: Changjoon Park

Student Number: K23030842

Degree Programme: MSc Cyber Security

Project Title: Explaining ML classifiers for malware detection

Supervisor: Fabio Pierazzi

Word count:

RELEASE OF PROJECT

Following the submission of your project, the Department would like to make it publicly available via the library electronic resources. You will retain copyright of the project.

I **agree** to the release of my project

I **do not** agree to the release of my project

Signature: Changjoon Park

Date: 21th April, 2024

INTRODUCTION

The accelerating evolution of malware poses significant threats to systems security, necessitating advanced defenses that not only detect but effectively understand malware mechanisms. Machine Learning (ML) has emerged as a cornerstone in crafting sophisticated malware detection systems. A machine learning-based malware analysis system has demonstrated high accuracy in classifying unknown malware and detecting new malware types, outperforming traditional signature-based methods. [1]

However, the complexity of ML models in malware detection raises concerns about their transparency and reliability, given that these models often function as black boxes with unclear decision-making processes. This lack of transparency can undermine trust in their effectiveness and accuracy. [2] One concern is that when a machine learning model is trained on datasets with large volumes containing redundant features, it can lead to overfitting because the model might learn to recognize these irrelevant features instead of generalizing from the essential ones. This overfitting results in the model performing poorly on new, unseen data, thereby reducing its overall accuracy.

This project will explore the use of existing explanation methods, such as SHAP and LIME, to enhance our understanding of the learning process within classifiers. By analyzing how these tools identify and evaluate the importance of features extracted and used in the dataset, we can more effectively discern and eliminate redundant features. This targeted feature selection is expected to reduce overfitting during model training, leading to more robust and accurate malware detection systems.

AIMS AND OBJECTIVES

This project is designed with two main objectives:

OBJECTIVE 1: TO OPTIMIZE FEATURE SETS FOR MALWARE CLASSIFICATION

The project will conduct a detailed investigation into feature selection for enhancing the performance of machine learning models in Android malware detection.

1-1. Algorithm Selection

Identify the most effective machine learning algorithms for Android malware detection through experimental comparisons.

1-2. Feature Set Comparison

Evaluate the performance of the API feature subset against the full feature set to determine the efficiency and accuracy of classifiers in various configurations.

1-3. Feature Analysis

Systematically assess individual features within the API feature set to establish their significance and contribution to the malware classification model's effectiveness.

Objective 2: To Evaluate the Impact of Explanation Methods on Algorithm

This objective aims to enhance the interpretability of machine learning models used in malware detection, which is vital for trust and reliability.

1-1. Explanation Method Analysis

Investigate and critically assess the capacity of existing explanation methods, such as SHAP and LIME, to provide transparency in the machine learning decision-making process.

1-2. Enhancement of Model Design

Determine how explanation methods can be integrated into the malware classification workflow to avoid overfitting and enhance the design and understanding of detection algorithms.

Domain and Data Set

The domain of this study is malware detection within Android operating systems, focusing specifically on API features. The dataset employed is derived from the DREBIN dataset, using features collected between 2014 to 2018 from various sources including the Google Play Store, alternative Chinese and Russian markets, and other sources like Android websites, malware forums, and security blogs.

This dataset comprises a significant number of Android applications, both benign and malicious, including 232,843 benign applications and 26,387 malware samples. It extracts features from Android applications using two primary sources: the *Android Manifest file (AndroidManifest.xml)* and the *disassembled code*. [3]

Features 1: from the Android Manifest File (AndroidManifest.xml)

1-1. Hardware Components

Features requested in the manifest that indicate access to hardware components such as the camera, GPS, microphone, and more. This set includes any hardware-related permissions that an application may need to function.

1-2. Requested Permissions

This set comprises permissions that the app requests during installation. These permissions are critical as they allow access to sensitive system resources and data.

1-3. App Components

This feature set includes the names of all activities, services, content providers, and broadcast receivers that are declared in the manifest of the application. These components play various roles in the app's operation, affecting how it interacts with the system and the user.

1-4. Filtered Intents

This set captures the intents listed in the manifest. Intents facilitate communication between components of the application and between different applications, which can be leveraged for malicious activities.

FEATURES2: FROM DISASSEMBLED CODE

2-1. Restricted API Calls

These are API calls that are protected by the Android permission system and are found in the application's dex code. Monitoring these API calls is crucial as they can perform actions that significantly impact the user's privacy and the device's security.

2-2. Used Permissions

This set involves a subset of permissions that are both requested in the manifest and actively used in the application code, showing the actual use versus just requested permissions.

2-3. Suspicious API Calls

This feature set includes API calls that access sensitive data or resources and are typically employed by malware, such as those for sending SMS, accessing user accounts, or manipulating user data.

2-4. Network Addresses

Includes all IP addresses, hostnames, and URLs extracted from the code that could potentially be associated with malicious activities like botnets or command and control servers.

Each of these feature sets is crucial in Drebin's analysis as they provide a comprehensive view of what an application does and how it interacts with the device and user data, which is used to determine whether it exhibits malicious behavior.

TECHNICAL SPECIFICATIONS AND METHODOLOGIES

The methodology underpinning this research project is a synthesis of comprehensive data analysis, machine learning model selection and evaluation, and interpretability through explanation methods. These components are vital in the development of a robust Android malware detection system.

DATA ANALYSIS AND VISUALIZATION

The project begins with an in-depth analysis and visualization of the prepared dataset, providing statistics on the samples and features, along with checks for data distribution. This phase is critical for understanding the landscape of the dataset and for ensuring the data's suitability for further processing and model training.

Machine Learning Models

A suite of machine learning algorithms will be evaluated, including:

- **Support Vector Machines (SVM)**: Both linear and kernel-based SVMs will be tested to establish a baseline of performance for malware classification tasks.
- **Random Forest**: An ensemble learning method known for its high accuracy and ability to run efficiently on large databases.
- **Naive Bayes**: A simple yet effective probabilistic classifier based on applying Bayes' theorem.
- **Deep Learning Models**: Convolutional Neural Networks (CNNs) will be explored for their capacity to capture spatial hierarchies in data and will be considered for their applicability to the features extracted from Android applications.

These models will be trained using both full and filtered datasets. The full dataset provides a broad perspective, while the filtered dataset focuses on API features—315 unique 'api_calls' features —to fine-tune and understand the relevance of specific feature types.

Model Evaluation

The models' performances will be rigorously evaluated using a variety of metrics:

- **Accuracy**: A straightforward metric to gauge the overall correctness of the model.
- **F1-Score**: A balanced measure that considers both precision and recall, particularly useful for unbalanced classes.
- **ROC & AUC**: Receiver Operating Characteristic curve and Area Under the Curve provide insights into the model's trade-offs between true positive rate and false positive rate.
- **Confusion Matrix**: Offers a detailed view of the classification results, allowing for the inspection of false positives and negatives.

Predictions and estimations on the expected performance of the models using the dataset will guide further tuning and adjustments.

Model Explanation with SHAP

The interpretability of machine learning decisions is crucial for their trustworthiness. SHAP (SHapley Additive exPlanations) will be utilized to provide explanations for model predictions, elucidating how each feature contributes to the output. This process will offer deeper insights into the model's behavior and help mitigate the issue of models taking "shortcuts" by relying on features that don't truly represent malicious behavior (shortcut learning).

Continuous Iteration and Improvement

The project adopts an iterative approach, where the feedback from model evaluation is used to refine and improve the methodologies. Machine learning models will be tuned, and their parameters adjusted according to the insights gained from SHAP explanations and evaluation metrics. This cyclical process ensures that the models evolve in response to new findings, leading to more sophisticated and accurate malware detection over time.

Background and Literature Review

State-of-the-Art in Android Malware Detection

The landscape of Android malware detection has dramatically evolved over the past decade, transitioning from traditional signature-based methods to sophisticated machine learning (ML) models. Signature-based approaches, while effective against known threats, struggle to keep pace with the rapid proliferation of malware variants and novel threats. This challenge has spurred the development of ML models that can learn from vast datasets to identify and categorize malware based on behavioral patterns rather than static signatures.

A seminal contribution to this field is encapsulated in the "DREBIN" paper, which introduces a lightweight method for detecting Android malware directly on smartphones. Unlike traditional methods, DREBIN performs a broad static analysis that collects extensive features from Android applications, including requested permissions, API calls, and network addresses. These features are then embedded into a vector space, enabling the application of ML techniques to detect malicious patterns automatically. Crucially, DREBIN not only identifies malware but also provides explanations for its detections, making its decisions transparent and understandable to users. [3]

Machine Learning Models in Malware Detection

Among the various ML models employed in malware detection, Linear Support Vector Machines (SVM) and Random Forests stand out due to their efficacy and efficiency. Linear SVM is particularly valued for its ability to handle high-dimensional data, as it constructs a hyperplane in a multidimensional space to differentiate between the classes (malicious vs. benign). This model is robust in environments with clear margin separation between classes and has been effectively used in systems like DREBIN.

On the other hand, Random Forests offer advantages in scenarios where the decision boundaries are more complex. As an ensemble learning method that builds multiple decision trees and merges them to get more accurate and stable predictions, Random Forests are a consistent and adaptable learning algorithm for classifying non-linear data. [4] They are particularly useful in malware detection for their ability to perform feature selection and handle unbalanced data, which is common in cybersecurity threats. In experimental evaluations, Random Forest achieved a high accuracy rate of 99.78% when combined with Variance Threshold feature reduction, significantly outperforming deep learning models which reached accuracy rates of around 98.99%. [5] This demonstrates Random Forest's robustness in complex malware detection scenarios, excelling in environments with intricate decision-making requirements.

Explanation Methods in Machine Learning

As ML models have become more prevalent in high-stakes domains like cybersecurity, the need for explainability in these models has grown. Explanation methods such as SHAP (SHapley Additive exPlanations) and LIME (Local Interpretable Model-agnostic Explanations) address this need by providing insights into the decision-making processes of ML models. SHAP values interpret the impact of having a certain value for a given feature in comparison to the prediction we'd make if that feature

took some baseline value, thus offering an in-depth look at the contribution of each feature to the output of the model.

LIME complements this by approximating the local prediction boundary with an interpretable model, which can be linear or decision-tree-based. It perturbs the input data and observes the corresponding changes in the output, facilitating an understanding of which features significantly influence the output in a localized region around the instance being explained.

Critical Review of Literature

Machine learning models have revolutionized the approach to malware detection by focusing on behavioral patterns rather than static characteristics. A prime example of this evolution is the "DREBIN" approach, which leverages a broad static analysis to gather extensive application features from Android apps, such as used permissions, API calls, and network addresses. These features are processed in a machine learning framework to identify indicative patterns of malware.

"DREBIN" stands out for its effectiveness in real-time environments. In an evaluation involving over 120,000 applications, including more than 5,000 confirmed malware samples, "DREBIN" achieved a detection rate of 94% with a low false-positive rate of 1%. This performance significantly outstrips traditional antivirus solutions, which struggle with rapidly mutating malware signatures and often fail to detect new, unknown threats.

However, while the static analysis method used by "DREBIN" is powerful, it is not without limitations. Static methods can be evaded by sophisticated malware that employs dynamic code execution or obfuscation techniques, which are not detectable without executing the application. Despite these challenges, "DREBIN" enhances the transparency of its process through explainable detection, providing justifications for each of its detections, which is a considerable advantage over black-box models that offer no insight into their internal decision-making processes.

Novel Contributions of the "DREBIN" Methodology

The novel contributions of the "DREBIN" methodology are particularly noteworthy. First, its comprehensive feature extraction strategy that encompasses both manifest and code properties of Android applications allows for a granular analysis of potential threats. By embedding these features into a joint vector space, "DREBIN" facilitates the use of linear classifiers to efficiently identify malware, a method that has proven effective even on constrained devices like smartphones.

Furthermore, "DREBIN" introduces explainability to the realm of malware detection. Using techniques akin to those in the SHAP and LIME frameworks, "DREBIN" provides explanations for its detections by highlighting the specific application features that most contributed to its decision. This not only aids security analysts and researchers in understanding the basis of the detection but also builds trust among users by clarifying why certain applications are flagged as malicious.

Limitations and Areas for Improvement

While "DREBIN" and similar ML-based methods have significantly advanced Android malware detection, they also have limitations that must be addressed. The reliance on static features, while resource-efficient, misses dynamically loaded code and runtime behaviors that could signify

sophisticated malware attacks. Additionally, the static nature of the analysis may not keep pace with the rapid evolution of Android malware, which continuously develops new evasion techniques.

Moreover, the explainability provided, while pioneering, is often limited to the scope of the features considered in the analysis. As malware developers increasingly incorporate complex obfuscation and evasion techniques, the explanatory models may need to evolve to maintain their effectiveness and accuracy.

Previous Work on Machine Learning Models in Malware Detection

Traditional malware detection methods relied heavily on manual feature engineering, which involved domain experts identifying and codifying specific attributes or behaviors characteristic of malware. This approach was not only labor-intensive but also limited in scalability and adaptability to new malware threats.

The shift towards automated feature learning in ML models has significantly enhanced the ability to adapt to the constantly evolving landscape of malware. Techniques such as deep learning allow models to automatically identify relevant features from raw data without explicit programming. This shift is evident in the use of convolutional neural networks (CNNs) and recurrent neural networks (RNNs), which have been particularly effective in learning complex patterns in data sequences and images, applicable to the analysis of executable files.

Effective Machine Learning Models in Malware Detection

Several machine learning models have proven effective in the domain of malware detection, each offering unique advantages:

1-1. Support Vector Machines (SVM)

SVMs are particularly valued for their effectiveness in high-dimensional spaces, which is common in malware detection due to the large number of features involved. They work by finding the hyperplane that best separates different classes (e.g., malicious vs. benign software) with the maximum margin. This method is noted for its robustness in avoiding overfitting, making it suitable for environments where precision is critical.

1-2. Random Forests

As an ensemble learning technique, Random Forests build multiple decision trees and merge them to improve the accuracy and avoid the overfitting problem of single decision trees. Each tree in the forest considers a random subset of features when forming decisions, which enhances the generalizability of the model across diverse types of malwares.

1-3. Deep Learning Models

Recent advances have included the use of deep learning architectures, which are particularly adept at handling unstructured data that is typical in malware analysis. Models such as deep neural networks (DNNs), CNNs, and autoencoders have been used to capture intricate patterns

and anomalies in application code, significantly improving detection rates, especially for zero-day threats.

While the "DREBIN" paper focuses on a linear SVM approach for its efficiency and effectiveness in real-time detection on smartphones, it is complemented by the broad static analysis which allows for a comprehensive feature extraction directly influencing the SVM's performance. In comparison, deep learning models, while highly effective, often require more computational resources and may not be as feasible for on-device analysis in real-time scenarios. Random Forests, being less resource-intensive than deep learning models, provide a middle ground by offering robust detection capabilities without the high computational overhead.

Critique of Existing Methods

In the field of Android malware detection using machine learning, existing methods have demonstrated significant strides in identifying and categorizing potential threats. However, despite these advancements, several critical challenges remain, particularly concerning the gap between laboratory model performance and real-world applicability, model overfitting, transparency in decision-making, and the handling of zero-day malware. This section critically assesses these issues and evaluates how methods like those presented in the "DREBIN" paper fit within this context.

Overfitting and Transparency in Decision-Making

Overfitting remains a pervasive issue in ML applications, where a model performs well on training data but fails to generalize to unseen data. This is particularly problematic in malware detection, where the ability to generalize is crucial due to the constantly evolving nature of threats. Techniques like cross-validation and regularization are commonly used to mitigate overfitting, but they do not address the fundamental issue of model obsolescence as new malware types emerge.

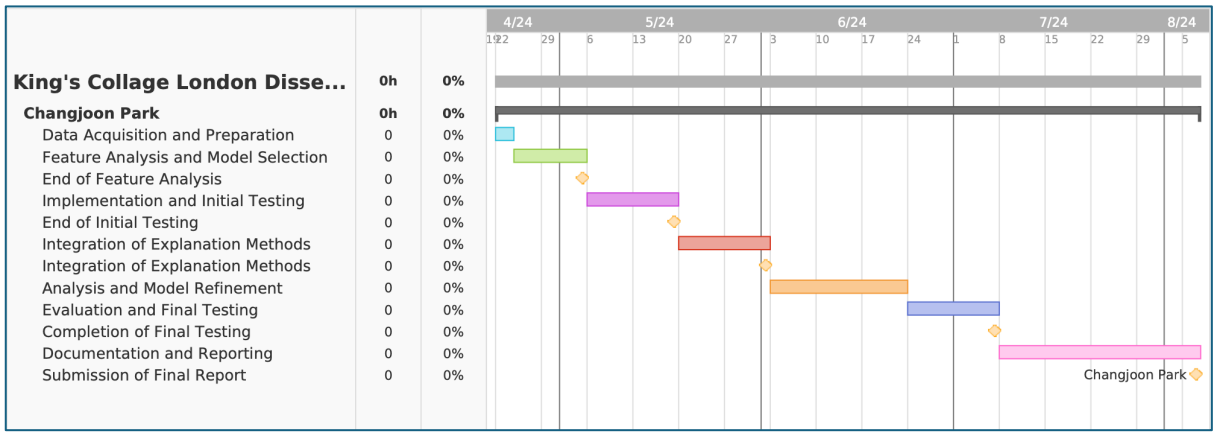
Moreover, the transparency of decision-making processes in ML models is critical for trust and verifiability, especially in security applications. Many advanced ML models, particularly those involving deep learning, act as "black boxes," where the decision processes are not easily interpretable. "DREBIN" significantly contributes to this area by providing explanations for its classifications, linking detected features directly to their impact on the decision, thus enhancing transparency and user trust.

Handling Zero-Day Malware

Zero-day malware, which exploits previously unknown vulnerabilities, presents a unique challenge. By definition, zero-day attacks are not present in the training data used for traditional ML models, making them particularly difficult to detect. This underscores the need for models that do not solely rely on historical data but can also detect anomalies and patterns indicative of novel attack strategies.

While "DREBIN" utilizes a form of anomaly detection by analyzing deviations in feature patterns, its reliance on static analysis may limit its effectiveness against sophisticated zero-day threats that employ dynamic execution techniques to evade detection. Here, the integration of dynamic behavioral analysis could complement "DREBIN's" static approach, providing a more robust defense against these advanced threats.

PROJECT SCHEDULE



PROJECT PHASES OVERVIEW

1. **Data Acquisition and Preparation (- 24th April 2024)**
   - Acquire the DREBIN dataset.
   - Perform initial data cleaning and preprocessing to ensure data quality and usability.

2. **Feature Analysis and Model Selection (24th April 2024 – 6th May 2024)**
   - Conduct detailed feature analysis to identify relevant API features.
   - Select and configure initial machine learning models (SVM, Random Forest, Deep Learning models).

3. **Implementation and Initial Testing (6th May 2024 – 20th May 2024)**
   - Implement the chosen models using the selected features.
   - Perform initial tests to evaluate model accuracy and identify areas of potential overfitting.

4. **Integration of Explanation Methods (20th May 2024 – 3rd June 2024)**
   - Integrate SHAP for model explanation.
   - Adjust models based on insights from explanation methods to improve transparency and reduce overfitting.

5. **Analysis and Model Refinement (3rd June 2024 – 24th June 2024)**
   - Analyze the output from the explanation methods to refine the feature set and model parameters.
   - Enhance model designs to better handle unseen data and reduce false positives.

6. **Evaluation and Final Testing (24th June 2024 - 8th July 2024)**
   - Conduct comprehensive testing across various metrics (accuracy, F1-score, ROC & AUC).
   - Final adjustments and tuning of models based on test results.

7. **Documentation and Reporting (8th July 2024 - 6th August 2024)**
    - Compile findings, methodologies, and results into the final project report.
    - Prepare presentation for project defense.

MILESTONES

1. **End of Feature Analysis (6th May 2024)**
    - Completion of feature set comparison and initial algorithm performance review.

2. **End of Initial Testing (20th May 2024)**
    - Initial model evaluations and identification of key areas for improvement.

3. **Integration of Explanation Methods (3rd June 2024)**
    - Successful integration of SHAP into the ML workflow.

4. **End of Model Refinement (24th June 2024)**
    - Finalization of model adjustments based on explanation insights.

5. **Completion of Final Testing (8th July 2024)**
    - Final model performance validation.

6. **Submission of Final Report (6th August 2024)**
    - Completion and submission of the comprehensive final report.

BIBLIOGRAPHY

[1] L. Liu, W. Baosheng, Y. Bo and Z. Qiuxi, "Automatic malware classification and new malware detection using machine learning," *Frontiers of Information Technology & Electronic Engineering,* vol. 18, pp. 1336-1347, 2017.

[2] G. Daniel, M. Carles and P. Jordi, "The rise of machine learning for detection and classification of malware: Research developments, trends and challenges," *J. Netw. Comput. Appl.,* vol. 153, p. 102526, 2020.

[3] A. Dan, S. Michael, H. Malte, G. Hugo and R. Konrad, "DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket," in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2014.

[4] S. Erwan and J.-P. V. Biau, "Consistency of Random Forests," *Annals of Statistics,* vol. 43, pp. 1716-1741, 2014.

[5] R. Hemant, A. Swati and S. Mohit, "Malware Detection Using Machine Learning and Deep Learning," in *Lecture Notes in Computer Science*, Springer International Publishing, 2018, p. 402–411.