AI-Powered Detection of Suspicious PowerShell Scripts Using NLP-Based Features

Introduction

Advanced mechanisms are now required to secure systems and efficiently reduce risks due to the rapid evolution of cyber threats. PowerShell is a robust scripting language and tool frequently used for automation and system administration, but it has two sides. Because of its adaptability and inherent access to vital system resources, it has become a preferred tool for attackers even if it helps administrators improve efficiency and streamline work. PowerShell is a significant attack vector in contemporary cybersecurity because of its dual nature.

Traditional signature-based detection techniques are practical against known threats but frequently fail to detect new or disguised dangerous programs. Advanced obfuscation techniques are commonly used by attackers, making traditional detection methods insufficient. Therefore, there is a pressing need for clever and flexible solutions to keep up with the ever-evolving threat landscape.

The goal of this project, "AI-Powered Detection of Suspicious PowerShell Scripts Using NLP-Based Features," is to improve the detection of malicious PowerShell scripts by utilising artificial intelligence (AI). The suggested method analyses and categorises scripts using Natural Language Processing (NLP) features, offering a reliable and scalable solution that can be adjusted to meet new threats.

This project aims to create an AI-powered system that can efficiently detect malicious activity by studying PowerShell scripts. The system extracts pertinent features using NLP-based techniques to train a machine-learning model. As a result, cybersecurity defences against PowerShell-based malware and attack vectors are strengthened by an automated, effective, and flexible solution.

This work is essential because it can potentially transform threat detection systems by resolving the drawbacks of conventional techniques. This initiative enhances the precision and versatility of malicious script detection, which advances the larger objective of protecting systems against changing cyber threats.

Literature Review

Malicious PowerShell script detection has drawn more attention in the cybersecurity community, and several approaches have been put out to deal with the difficulties caused by the obfuscation and adaptability of malicious scripts. This section examines important works that advance this field and identifies the gaps that these works left, which this initiative seeks to fill.

1. "Effective Method for Detecting Malicious PowerShell Scripts Based on Hybrid Features"

In order to categorise PowerShell scripts, this paper suggests a detection model that integrates static, dynamic, and semantic data, such as textual tokens, FastText embeddings, and abstract syntax tree (AST) analysis. The model's accuracy on both harmful and benign scripts was 98.93% using a Random Forest classifier. The hybrid solution uses contextual and structural elements to handle both mixed-content and obfuscated scripts efficiently.

- **Strengths**: High accuracy and ability to analyse complex scripts.
- **Limitations**: Dependency on time-consuming static and dynamic analysis steps, making it less suitable for real-time or lightweight applications.

2. "Detecting Malicious PowerShell Commands Using Deep Neural Networks"

This study examines the application of Deep Neural Networks (DNNs) in conjunction with conventional NLP methods to identify harmful PowerShell commands. The researchers showed that the integration of character-level convolutional neural networks (CNNs) and NLP-based classifiers achieves excellent recall and precision, even in the case of obfuscated scripts. The ensemble methodology overcomes the shortcomings of standalone models by recognising various patterns within malicious scripts.

- **Strengths**: High performance in detecting obfuscated commands.
- **Limitations**: Relies heavily on ensemble methods, which can increase computational overhead. Does not focus on standalone NLP-based feature extraction for lightweight analysis.

3. "Static Detection of Malicious PowerShell Based on Word Embeddings"

This study presents a static detection technique that utilises word embeddings such as Doc2Vec for efficient examination of PowerShell scripts. The method focuses on reproducibility by employing datasets that are publicly accessible. The model attained an F1 score of 0.995 in real-world scenarios, demonstrating its efficacy against emerging malware families.

- **Strengths**: Avoids reliance on dynamic analysis, providing a more efficient and reproducible method.
- **Limitations**: May struggle with deeply obfuscated scripts where contextual relationships are deliberately obscured.

Identified Gaps in Existing Research

1. Dependency on Dynamic Analysis: Many approaches rely on dynamic features or hybrid techniques, which are computationally expensive and time-consuming.
2. Complex Model Architectures: Ensemble and hybrid models, while accurate, may not be suitable for real-time applications due to their computational demands.
3. Reproducibility Challenges: Private datasets or lack of transparency in preprocessing steps make reproducing results difficult.

Relevance to This Project

The suggested method in this project aims to fill the identified gaps by concentrating exclusively on NLP-based characteristics derived from PowerShell scripts. By utilising word embeddings and language models, this research seeks to offer a solution that is efficient, scalable, and reproducible while circumventing the complexities linked to static and dynamic analysis. This renders the proposed technique suitable for lightweight applications and the real-time identification of potentially harmful scripts.

Proposed Methodology

The suggested methodology aims to utilise techniques based on Natural Language Processing (NLP) to identify harmful PowerShell scripts. This strategy includes extracting text-based features and training models and ensuring practical application via a web-based platform and PowerShell integration.

1. Data Preprocessing

- **Normalization**: Convert scripts to lowercase and standardise formatting to handle obfuscation.
- **Tokenization**: Split scripts into meaningful units such as cmdlets, keywords, and parameters.
- **Stopword Removal**: Remove non-essential PowerShell keywords and characters that do not contribute to malicious behaviour detection.
- **Placeholder Replacement**: Replace sensitive elements like IP addresses, URLs, and file paths with generic placeholders (e.g., <IP_ADDR>).

2. Feature Extraction

- **TF-IDF Vectorization**:
    - Scripts are represented using Term Frequency-Inverse Document Frequency (TF-IDF), which assigns weights to terms based on their importance across the dataset.
    - This captures the contextual significance of words within scripts, making it an efficient NLP-based feature extraction technique.

3. Model Training

- Multiple machine learning classifiers are trained and evaluated using the extracted TF-IDF features:
    - Random Forest: A robust ensemble model that performs well with text classification tasks.
    - Support Vector Machine (SVM): A classifier optimized for handling high-dimensional text data.
    - Logistic Regression: A baseline linear classifier to compare performance.
- **Evaluation Metrics**:
    - Models are assessed using standard metrics, including accuracy, precision, recall, and F1-score.

- Confusion matrices and feature importance are analysed to interpret results effectively.

4. Real-World Integration

- **Streamlit Application**:
  - A user-friendly interface allows users to upload .ps1 files for analysis.
  - The app outputs the classification result (benign or malicious) along with the probability scores for transparency.
- **PowerShell Integration**:
  - A real-time command line interface processes single-line PowerShell scripts.
  - The system provides immediate feedback on whether the script is malicious and prompts the user to confirm or cancel execution if flagged as harmful.

5. Workflow Overview

i) User uploads or inputs a PowerShell script.
ii) Preprocessing cleans and standardises the script.
iii) Features are extracted using TF-IDF vectorisation.
iv) The trained machine learning model analyses the script and classifies it as benign or malicious.
v) Results are displayed in the Streamlit app or PowerShell interface, ensuring real-world applicability.

Implementation

The implementation of the proposed system for detecting malicious PowerShell scripts was performed in Python, incorporating feature extraction, model training, and real-world usability through a Streamlit app and PowerShell integration.

1. Feature Extraction and Model Training

1) Dataset
    - The dataset comprises PowerShell scripts categorised into benign, mixed, and malicious.
    - Scripts are pre-processed to:
        - Normalize data by converting text to lowercase.
        - Remove comments and unnecessary characters such as punctuation.
        - Tokenize content into meaningful units for analysis.
2) Feature Extraction
    - TF-IDF Vectorization is used to transform scripts into numerical representations by assigning importance scores to terms based on their frequency within the dataset.
    - Focus is placed on extracting term relationships and contextual importance using this NLP-based technique.
3) Model Training
    - Multiple machine learning classifiers are trained using Scikit-learn, including:
        - Random Forest Classifier for robust performance.
        - Logistic Regression as a baseline model.
        - Support Vector Machine (SVM) for handling high-dimensional text data.
    - Cross-validation ensures the reliability and generalizability of the models.
    - Performance is validated using metrics such as:
        - Accuracy
        - Precision
        - Recall
        - F1-score
    - Confusion matrices and feature importance are used to interpret the models' performance.

2. Streamlit Application

- A Streamlit-based web interface is developed to facilitate user interaction:
    - **Upload Functionality**: Users can upload .ps1 files for classification.
    - **Prediction Output**:
        - Displays whether a script is benign or malicious.

        - Provides probability scores for the classification, ensuring transparency in predictions.

3. PowerShell Integration

- A **PowerShell command-line interface** is created to analyse and classify single-line PowerShell scripts in real-time:
    - Predicts the script's status (benign or malicious).
    - Outputs the probability scores for each classification.
    - If the script is flagged as malicious, the user is prompted to confirm or cancel its execution, enhancing security without impacting usability.

4. Tools and Libraries

- **Programming Language**: Python
- **Libraries Used**:
    - **Scikit-learn**: For feature extraction, model training, and evaluation.
    - **NLTK/spaCy**: For preprocessing and tokenisation.
    - **Streamlit**: To create a user-friendly web application.

Results

1. Model Comparison

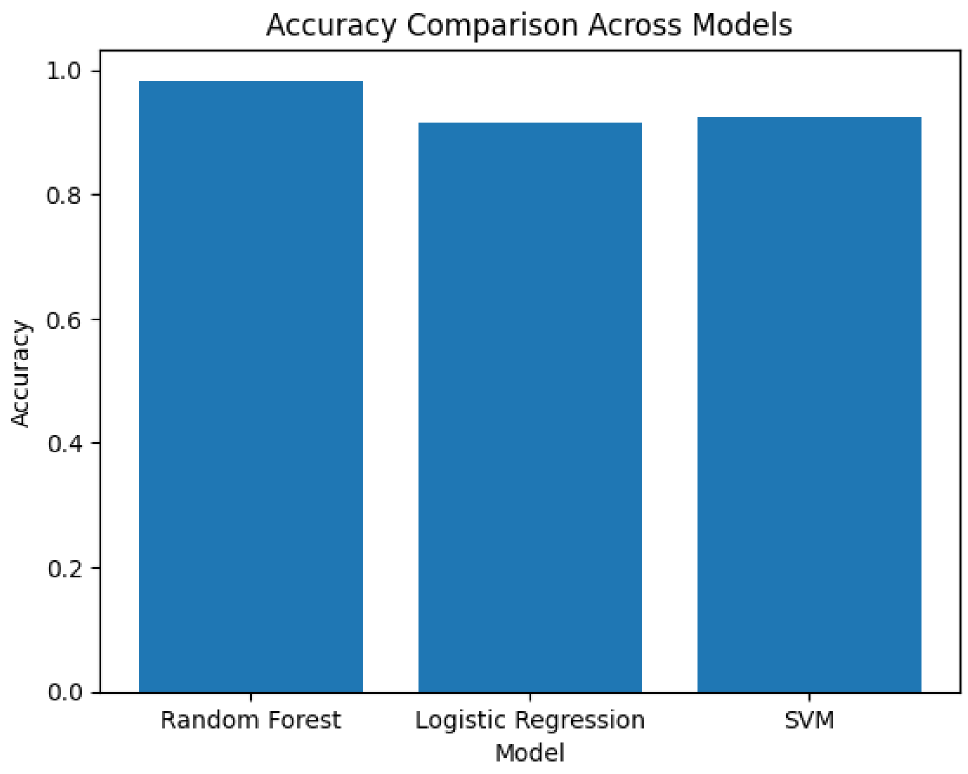
- The results of the proposed methodology are summarised below, showcasing the performance metrics for three classifiers:

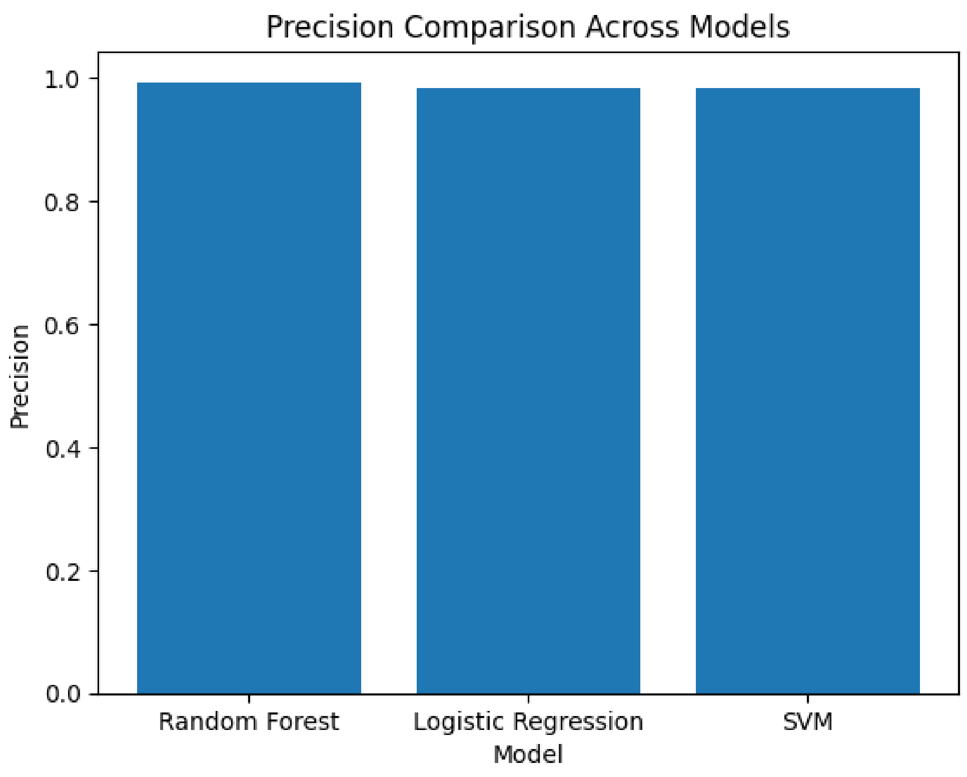
Model	Accuracy	Precision	Recall	F1 Score
Random Forest	0.983	0.993	0.982	0.987
Logistic Regression	0.915	0.984	0.887	0.933
Support Vector Machine (SVM)	0.923	0.983	0.899	0.939

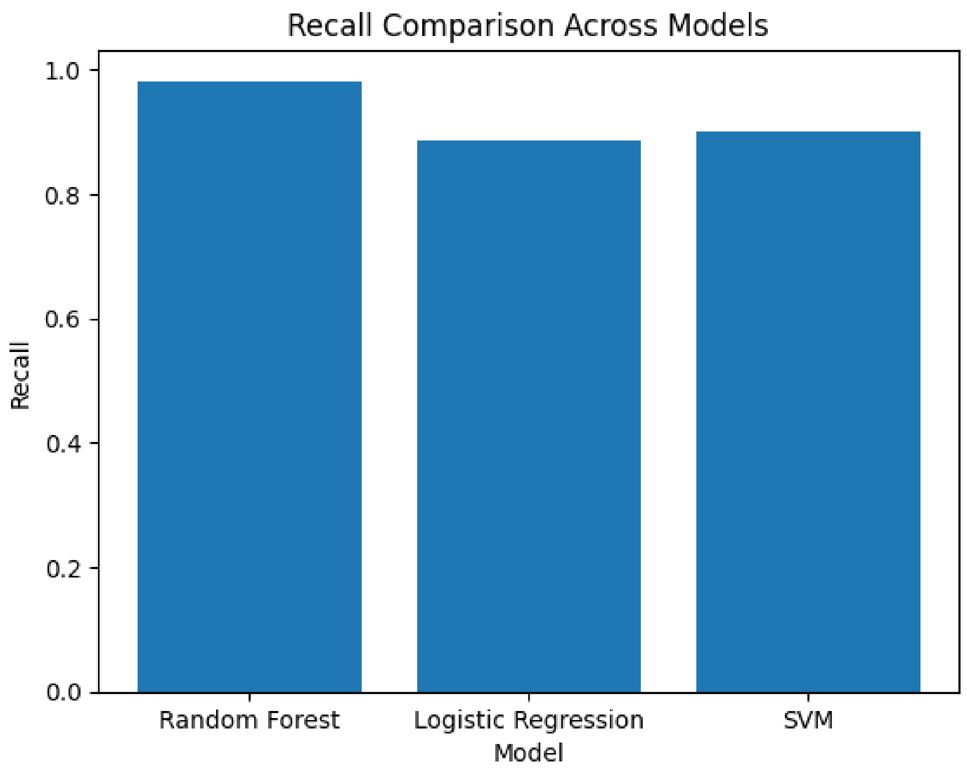
2. Baseline Comparison

The baseline model from the primary paper, **"Effective Method for Detecting Malicious PowerShell Scripts Based on Hybrid Features"**, achieved a detection accuracy of **98.93%** by using a combination of static, dynamic, and semantic features. In contrast:

- The **Random Forest classifier** achieves comparable accuracy (98.3%) using only NLP-based features, without the overhead of static or dynamic analysis.
- The **F1-score** of the Random Forest model is also competitive (0.987), demonstrating the robustness of the proposed lightweight approach.




Precision Comparison Across Models
Precision
1.0
0.8
0.6
0.4
0.2
0.0
Random Forest
Logistic Regression
SVM
Model


Recall Comparison Across Models
Recall
1.0
0.8
0.6
0.4
0.2
0.0
Random Forest
Logistic Regression
SVM
Model

Discussion

1. Interpretation of Results

The results demonstrate the efficiency of NLP-based feature extraction for detecting malicious PowerShell scripts:

- The Random Forest Classifier achieved the highest accuracy (98.3%) and F1-score (0.987), making it the most robust model among those tested.
- Both Logistic Regression and SVM also performed well, with F1-scores of 0.933 and 0.939, respectively, validating the effectiveness of NLP-based techniques for this task.

In contrast to the baseline method presented in the main paper, which reached an accuracy of 98.93% through a hybrid strategy (incorporating static, dynamic, and semantic features), the proposed approach attains a comparable accuracy by utilising only lightweight NLP-based features. This underscores the capability of NLP techniques to deliver similar performance while considerably minimising computational demands.

2. Strengths

- **Efficiency**: By relying on NLP-based features like TF-IDF, the approach eliminates the need for static or dynamic analysis, making it computationally efficient and suitable for real-time applications.
- **Scalability**: The methodology can handle large datasets and diverse PowerShell scripts, as demonstrated by its high accuracy across different classifiers.
- **Usability**: The integration of a Streamlit application and PowerShell interface enhances the practicality of the solution, making it accessible to system administrators and cybersecurity professionals.
- **Performance**: The Random Forest Classifier's high F1-score (0.987) ensures a balance between precision and recall, reducing the likelihood of false positives or negatives.

3. Limitations

- **Obfuscated Scripts**: While NLP-based features perform well in general, deeply obfuscated scripts may still pose challenges, especially if the tokenised content lacks meaningful patterns.
- **Feature Dependency**: The reliance on TF-IDF might limit the ability to capture deeper contextual relationships or dependencies present in highly complex scripts. Advanced techniques like deep learning-based embeddings (e.g., BERT) could address this limitation.

- **Comparative Evaluation**: Although the results are strong, a more direct evaluation against the baseline method (e.g., testing on the same dataset or reproducing their hybrid feature model) could provide a clearer comparison.

4. Future Directions

To build upon this work, the following areas could be explored:

- **Enhanced Feature Extraction**: Incorporating advanced NLP models like Transformers or embeddings such as BERT to capture deeper semantic and contextual relationships.
- **Dataset Expansion**: Extending the dataset to include more diverse and obfuscated malicious scripts to improve robustness.
- **Comprehensive Comparison**: Reproducing the baseline method to evaluate the relative advantages of NLP-based features in greater detail.
- **Adaptive Models**: Exploring online learning techniques to allow the model to adapt dynamically to new threats and patterns.

Conclusion

This project successfully developed an efficient and effective system for detecting malicious PowerShell scripts using Natural Language Processing (NLP)-based features. By using methods such as TF-IDF for extracting features and implementing machine learning algorithms like Random Forest, Logistic Regression, and Support Vector Machines, the proposed approach demonstrated impressive accuracy and dependability.

Key Contributions

1. **High Performance**: The Random Forest classifier achieved an accuracy of 98.3% and an F1-score of 0.987, demonstrating that NLP-based features can match the performance of more computationally intensive hybrid approaches.
2. **Efficiency**: By eliminating the need for static and dynamic analysis, the system reduces computational overhead, making it suitable for real-time applications.
3. **Usability**: The integration of a Streamlit application and PowerShell interface ensures practical applicability, providing users with an accessible and interactive tool for script analysis.
4. **Broader Impact**: This work advances the state of malware detection by offering a lightweight and scalable solution that addresses the challenges of obfuscated and evolving malicious scripts.

Future Work

To further enhance the system's capabilities, the following directions are proposed:

1. **Advanced NLP Models**: Incorporating deep learning techniques such as Transformers (e.g., BERT) to capture richer semantic relationships in script content.

2. **Enhanced Dataset**: Expanding the dataset to include a wider variety of obfuscated and complex malicious scripts to improve generalisation.

3. **Dynamic Threat Adaptation**: Developing adaptive learning models that continuously evolve to recognise emerging threats.

4. **Comprehensive Evaluation**: Conducting a direct comparison with existing state-of-the-art hybrid methods to validate the scalability and robustness of the proposed approach.

In summary, this project illustrates that techniques relying on NLP provide an appealing substitute for conventional malware detection methods, merging exceptional accuracy with computational effectiveness. It establishes a foundation for upcoming research and real-world applications focused on enhancing defences against threats stemming from PowerShell.

References

1. Alaidaros, A., Darabkh, K. A., & Almomani, A. (2023). *Effective Method for Detecting Malicious PowerShell Scripts Based on Hybrid Features*. Journal of Information Security and Applications, 73, 103417.
2. Al-Qassas, R., & Darwish, O. (2022). *Detecting Malicious PowerShell Commands Using Deep Neural Networks*. International Journal of Computer Applications,
3. Vidhyalakshmi, S., & Manoharan, R. (2021). *Detecting Malicious and Clean PowerShell Scripts Among Obfuscated Commands Using Deep Learning Methods and Word Embedding*. Procedia Computer Science, 179, 733–741.
4. *Dataset*: Malicious PowerShell Dataset (MPSD). Retrieved from the GitHub repository.