A Multi-Judge System for Static Spyware Detection Using Machine Learning and Deep Learning

Trifan Bogdan-Cristian
trifan.bogdan09@yahoo.com

***Abstract*—This study proposes a multi-judge (ensemble) system for static detection of spyware in Windows executable files. The system combines five diverse classifiers, including classical machine learning algorithms (Logistic Regression, SVM, Random Forest and XGBoost) and deep learning models (1D CNN), each leveraging different representations of Portable Executable (PE) files.**

I. Introduction

Spyware is a prevalent form of malware that clandestinely collects user data and poses significant risks to privacy and security. While dynamic analysis techniques can be powerful, they are increasingly circumvented by anti-analysis and anti-VM techniques. Static analysis, which examines files without execution, offers a safer and more scalable solution. However, it requires robust feature engineering and modeling to counteract sophisticated obfuscation.

This study introduces a multi-judge system that integrates several machine learning and deep learning models, each utilizing distinct feature extraction strategies, to perform high-accuracy static spyware detection. This approach seeks to leverage the complementary strengths of different classifiers and feature spaces.

II. Computer limitations

Each model has been trained on a computer with the following specifications:

- **CPU**: Intel® CoreTM i5-12450H
- **GPU**: GeForce RTX 3050 4GB
- **RAM**: 16GB DDR4
- **Storage**: SSD

III. System Architecture and Methodology

A. Overview of the Multi-Judge System

The proposed system is composed of five classifiers, or "judges", each trained on a different view or representation of the PE files:

- **Logistic Regression, SVM, Random Forest and XGBoost**: Trained on engineered PE structural features.
- **Convolutional Neural Networks (CNNs)**: 1D CNN trained on raw byte sequences.

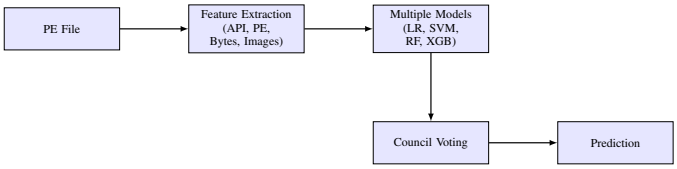


Fig. 1. Pipeline diagram for the multi-judge spyware detection system.

B. Dataset

The dataset used in this study consists of a total of 18,499 Windows Portable Executable (PE) files, divided into 9,998 spyware samples and 8,501 non-spyware samples. The non-spyware set includes both benign files and other malware types that are not categorized as spyware. The dataset was split as follows:

- **Training**: 6,998 spyware and 5,951 non-spyware samples
- **Validation**: 1,500 spyware and 1,275 non-spyware samples
- **Test**: 1,500 spyware and 1,275 non-spyware samples

The samples were collected from the following sources:

- **Spyware and malware**: Downloaded from https://virusshare.com/
- **Benign files**:
  - Selected samples from https://github.com/iosifache/DikeDataset
  - Selected samples from https://github.com/bormaa/Benign-NET
  - Personal benign binaries including Steam game executables, open-source Windows software, and system files
  - Portable software from https://portableapps.com/

C. Feature Extraction

To effectively capture both structural and behavioral characteristics of PE (Portable Executable) files, multiple complementary feature types were extracted:

- **PE Header and Structural Features**: These include various metadata fields and statistics derived from the PE headers and sections:
  - Raw header fields: number of sections, symbol count, image size, timestamp, entry point, checksum, machine type.
  - Linker version scores (major and minor), derived from empirical distributions.

  - Section statistics: average section size ratio, entropy stats (high-entropy section count), executable/writable flag counts, and entropy for specific sections (e.g., `.text`, `.rdata`, `.rsrc`, etc.).
  - Overlay size, presence of debug symbols, alignment fields, and TLS callback analysis.
- **API-Level Behavioral Features**:
  - Count of suspicious or spyware-related imported API calls (e.g., `CreateRemoteThread`, `RegSetValueExA`, `LoadLibraryA`, etc.).
  - Number of imported DLLs and total API functions.
  - Exported symbols and resource directory statistics (resource count, size, presence of icons/version info).
- **Section Name Anomalies**:
  - Sections with known suspicious names (blacklist hits), high-entropy names, or fuzzy matches (Levenshtein distance) to malware-related section names.
- **DLL Characteristics and Security Flags**:
  - ASLR, DEP/NX, high entropy VA, and guard flag presence are extracted from the `DllCharacteristics` field.
- **Raw Byte Features for CNN**:
  - A fixed-length (30,000-byte) raw byte sequence from the executable is extracted and passed into a 1D CNN model.

To maximize the performance, each model has been trained on different features that we're selected by applying ablation.

IV. Performance of Traditional Classifiers

To evaluate the effectiveness of each model independently, we trained and tested five classifiers using their respective optimized feature sets. The table below summarizes the performance of each classifier in terms of accuracy, precision, recall, and F1-score for both non-spyware (class 0) and spyware (class 1) samples.

TABLE I
Performance of Classical Models

Model	Accuracy	Precision (0/1)	Recall (0/1)	F1 (0/1)
Logistic Regression	94,7%	95,4% / 94,2%	93% / 96,2%	94,2% / 95,1%
SVM	94,5%	94,3% / 94,7%	93,7% / 95,2%	94% / 95%
Random Forest	98%	96,4% / 99,2%	99,1% / 96,8%	97,7% / 98%
XGBoost	98%	97,4% / 98,6%	98,3% / 97,8%	97,9% / 98,1%
1D CNN	94%	93% / 95%	94% / 94%	94% / 95%

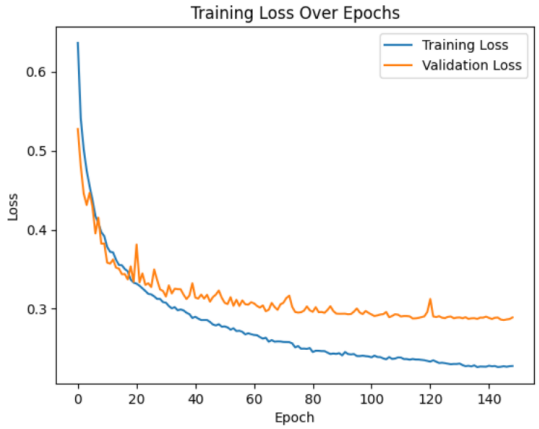
V. Deep Learning Architectures and Results

A. 1D CNN on Raw Byte Sequences

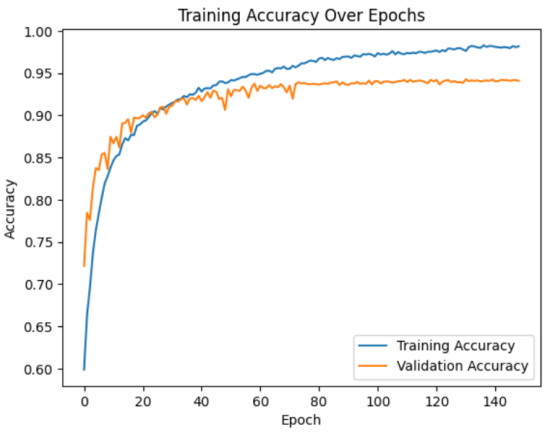
The 1D CNN was trained directly on raw byte sequences. Each input was a sequence of 30,000 bytes (files were padded as needed) embedded into 16-dimensional vectors. The network architecture comprised two convolutional layers with batch normalization, ReLU activations, dropout, and max-pooling.

TABLE II
1D CNN Architecture Summary

Layer Type	Parameters
`Conv1d`	in = 16, out = 32, kernel = 7, stride = 2, padding = 3
`BatchNorm1d`	in = 32
`ReLU`	—
`MaxPool1d`	kernel = 2, stride = 2
`Dropout1d`	p = 0.3
`Conv1d`	in = 32, out = 64, kernel = 5, stride = 1, padding = 2
`BatchNorm1d`	in = 64
`ReLU`	—
`MaxPool1d`	kernel = 2, stride = 2
`Dropout1d`	p = 0.3
`AdaptiveAvgPool1d`	out = 32
`Linear`	in = 64 32, out = 128
`BatchNorm1d`	in = 128
`ReLU`	—
`Dropout`	p = 0.5
`Linear`	in = 128, out = 2



(a) Training and validation loss per epoch



(b) Training and validation accuracy per epoch

Fig. 2. Performance curves for the 1D CNN model. (a) shows the loss values per epoch, while (b) shows the classification accuracy.

The training and validation loss curves in Fig. 2(a) indicate that the model learns effectively during the early epochs, with only mild overfitting observed beyond epoch 50. Early stopping was considered to prevent further divergence and maintain generalization.

TABLE III
TRAINING HYPERPARAMETERS USED FOR THE 1D CNN MODEL

Parameter	Value
Optimizer	AdamW
Learning rate	10^{-3}
Weight decay	10^{-4}
Batch size	512
Max epochs	200
Early stopping patience	12
Learning rate scheduler	ReduceLROnPlateau
Scheduler patience	8
LR reducing factor	0.5
Input sequence length	30,000 bytes

VI. ENSEMBLE VOTING (COUNCIL SYSTEM)

The final detection system employs an ensemble of five independently trained classifiers, referred to as the *Council*. For each input sample, every judge (model) produces a confidence score indicating the probability that the file is spyware. These individual scores are then combined into a single consensus score through a weighted voting mechanism.

Each judge is assigned a weight proportional to its standalone validation accuracy. The weights are defined as follows:

- Logistic Regression: 95.42
- Random Forest: 97.91
- Support Vector Machine: 95.39
- XGBoost: 98.78
- 1D Convolutional Neural Network: 94.09

To ensure fairness, the raw weights are normalized such that their sum equals 1:

$$w_i' = \frac{w_i}{\sum_{j=1}^{5} w_j}$$

where w_i is the raw accuracy-based weight of model i, and w_i' is the normalized weight.

The final ensemble confidence score S is then computed as:

$$S = \sum_{i=1}^{5} w_i' \cdot p_i$$

where p_i is the predicted spyware probability from judge i.

To increase sensitivity, a lower classification threshold of **0.35** (instead of the standard 0.5) was chosen, making the model more "paranoid" and biased toward detecting potential threats, at the expense of a slightly higher false positive rate.

Test Set Performance:

- **Overall Accuracy:** 97%
- **Confusion Matrix:**

$$\begin{bmatrix} 1215 & 60 \\ 29 & 1471 \end{bmatrix}$$

TABLE IV
COUNCIL PERFORMANCE ON TEST SET

Class	Precision	Recall	F1-Score	Support
Benign (0)	0.98	0.95	0.96	1275
Spyware (1)	0.96	0.98	0.97	1500
Accuracy		**97.00%**		

VII. LIMITATIONS

While the proposed multi-judge system achieves high accuracy in static spyware detection, several limitations remain:

- **Static Analysis Evasion**: The models rely exclusively on static features. Advanced malware using packing, encryption, or obfuscation may evade detection if their extracted features resemble benign files.
- **Dataset Bias**: Although care was taken to assemble a diverse dataset, there may still be distribution shifts or bias not captured in the current sample, especially for rare or novel spyware variants.
- **Label Noise**: Some benign samples may contain undetected malware, or vice versa, due to imperfect source labeling, which can impact training and evaluation.
- **Computational Requirements**: The ensemble, especially the deep learning model, requires significant computational resources for both training and inference, which may not be suitable for low-resource environments.

Future work could address these issues by incorporating dynamic analysis, adversarial training, or transfer learning to improve generalization to novel threats.

VIII. CONCLUSION

This paper presented a multi-judge ensemble system for static spyware detection using a combination of feature-engineered machine learning models and deep learning architectures. The results demonstrate that leveraging complementary representations and models can significantly improve detection accuracy and robustness.