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 six diverse classifiers, including classical machine learning algorithms (Naive Bayes, SVM, KNN, Random Forest, XGBoost) and deep learning models (1D and 2D CNNs), each leveraging different representations of Portable Executable (PE) files. Experimental results show that the ensemble approach achieves robust and accurate spyware detection, with XGBoost attaining up to 97% accuracy and 1D CNNs demonstrating superior generalization compared to 2D CNNs. The findings highlight the benefits of multi-representation ensembles for static malware analysis.**

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 six classifiers, or "judges", each trained on a different view or representation of the PE files:

- **Naive Bayes (NB)**: Trained on API call frequency vectors.
- **SVM, KNN, Random Forest, XGBoost**: Trained on engineered PE structural features.
- **Convolutional Neural Networks (CNNs)**: 2D CNNs trained on image representations; 1D CNNs trained on raw byte sequences.

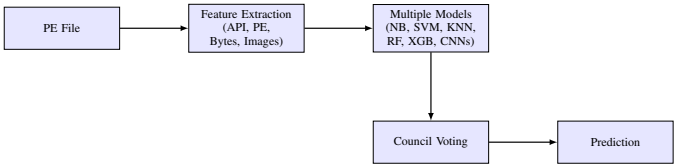


Fig. 1. Pipeline diagram for the multi-judge spyware detection system.

B. Dataset

The dataset used in this study consists of a total of 18,967 Windows Portable Executable (PE) files, divided into 10,000 spyware samples and 8,967 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**: 6,000 spyware and 5,380 non-spyware samples
- **Validation**: 2,000 spyware and 1,793 non-spyware samples
- **Test**: 2,000 spyware and 1,794 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 from Windows 10
  - Portable software from https://portableapps.com/

Duplicate files were removed, and samples were manually verified to avoid label noise.

C. Feature Extraction

To effectively capture both structural and semantic characteristics of PE files, multiple features were extracted:

- **API Call Features**: Counts of imported API symbols, normalized and used for Naive Bayes classification.

- **PE Header and Structural Features**: Custom-engineered features including:
  - Section count, imported DLLs and API functions (log-transformed)
  - Ordinal-only import presence, entropy statistics (mean, max, std)
  - Mean raw section size, executable/writable section counts
  - String statistics, exported symbol count, file age
  - Overlay size and presence, header flags (ASLR, NX), and 16-bin byte histograms
- **Byte Sequences**: Fixed-length (400,000 bytes) raw byte sequences for input to the 1D CNN.
- **Grayscale Images**: Each file converted into five 150×150 images (head, tail, and three entropy-selected regions) for 2D CNNs.

IV. Performance of Traditional Classifiers

TABLE I
PERFORMANCE OF CLASSICAL MODELS

Model	Accuracy	Precision	Recall	F1
NB (API)	0.83	0.86 / 0.82	0.78 / 0.89	0.82 / 0.85
SVM	0.95	0.94 / 0.96	0.96 / 0.94	0.95 / 0.95
KNN	0.95	0.95 / 0.96	0.96 / 0.95	0.95 / 0.96
Random Forest	0.96	0.94 / 0.99	0.98 / 0.95	0.96 / 0.97
XGBoost	0.97	0.96 / 0.98	0.98 / 0.96	0.97 / 0.97
1D CNN	0.884	0.883 / 0.886	0.871 / 0.896	0.877 / 0.891

As shown in Table 1, XGBoost achieved the highest accuracy (97%) among classical classifiers, followed by Random Forest, SVM, and KNN. Naive Bayes, using only API call frequencies, performed least effectively, indicating the need for richer feature sets.

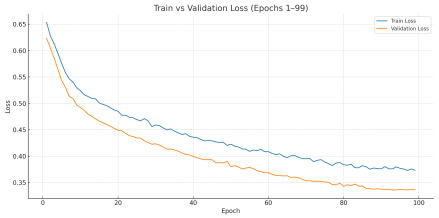
V. Deep Learning Architectures and Results

A. 2D CNNs on Image Representations

Each PE file was represented as five grayscale images. Three CNN models with similar architectures were trained with different loss functions and learning rates. All used Adam optimizer with a learning rate warmup and ReduceLROnPlateau scheduling.

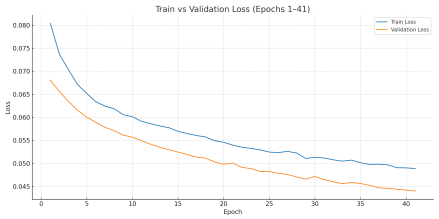
TABLE II
2D CNN MODEL RESULTS

Model	Loss	LR	Accuracy	Precision	F1-Score
M1	BCEWithLogits	2×10^{-5}	85.0%	86.7%	86.5%
M2	CrossEntropyLoss	5×10^{-5}	85.1%	86.1%	86.1%
M3	FocalLoss	2×10^{-5}	85.1%	86.1%	86.1%



(a) Training and validation loss per epoch (M1)

(b) Training and validation accuracy per epoch (M2)



(c) Training and validation accuracy per epoch (M3)

Fig. 2. Performance curves for the 2D CNN models

Alternative visual feature engineering, such as RGB images, byte histograms, Markov matrices, and Gabor textures, resulted in unstable training or poor convergence.

TABLE III
2D CNN ARCHITECTURE USED FOR CLASSIFICATION

Layer	Configuration	Output Shape
Input	5 grayscale images of 150×150	$5 \times 150 \times 150$
Conv2D	$5 \to 64$, kernel=3×3, stride=1, padding=1	$64 \times 150 \times 150$
BatchNorm2D	64 channels	$64 \times 150 \times 150$
ReLU	Activation	$64 \times 150 \times 150$
MaxPool2D	kernel=2×2, stride=2	$64 \times 75 \times 75$
Conv2D	$64 \to 128$, kernel=3×3, stride=1, padding=1	$128 \times 75 \times 75$
BatchNorm2D	128 channels	$128 \times 75 \times 75$
ReLU	Activation	$128 \times 75 \times 75$
Dropout2D	$p = 0.15$	$128 \times 75 \times 75$
MaxPool2D	kernel=2×2, stride=2	$128 \times 37 \times 37$
Conv2D	$128 \to 256$, kernel=3×3, stride=1, padding=1	$256 \times 37 \times 37$
BatchNorm2D	256 channels	$256 \times 37 \times 37$
ReLU	Activation	$256 \times 37 \times 37$
MaxPool2D	kernel=2×2, stride=2	$256 \times 18 \times 18$
AdaptiveAvgPool2D	Output size = 1×1	$256 \times 1 \times 1$
Flatten	—	256
Linear	$256 \to 32$	32
Dropout	$p = 0.5$	32
Linear	$32 \to 1$	1

B. 1D CNN on Raw Byte Sequences

The 1D CNN was trained directly on raw byte sequences. Each input was a sequence of 150,000 bytes (files were padded as needed) embedded into 64-dimensional vectors. The network architecture comprised three convolutional layers with

group normalization, GELU activations, dropout, and max-pooling. The classifier head consisted of two GELU-activated fully connected layers and a sigmoid output.

TABLE IV
1D CNN ARCHITECTURE SUMMARY

Layer Type	Parameters
`Conv1d`	in = 64, out = 128, kernel = 7, stride = 2, padding = 3
`GroupNorm`	8 groups, 128 channels
`GELU`	—
`Dropout1d`	p = 0.15
`MaxPool1d`	kernel = 2, stride = 2
`Conv1d`	in = 128, out = 256, kernel = 7, stride = 2, padding = 3
`GroupNorm`	8 groups, 256 channels
`GELU`	—
`Dropout1d`	p = 0.20
`Conv1d`	in = 256, out = 512, kernel = 7, stride = 1, padding = 6, dilation = 2
`GroupNorm`	8 groups, 512 channels
`GELU`	—
`Dropout1d`	p = 0.20
`MaxPool1d`	kernel = 2, stride = 2
`Linear`	in = 1024, out = 256
`GELU`	—
`Dropout`	p = 0.35
`Linear`	in = 256, out = 32
`GELU`	—
`Dropout`	p = 0.15
`Linear`	in = 32, out = 1

AdamW was used for optimization, with learning rate scheduling and gradient accumulation to enable large batch sizes on limited GPU memory. Binary cross-entropy loss was used.

TABLE V
TRAINING HYPERPARAMETERS USED FOR THE 1D CNN MODEL

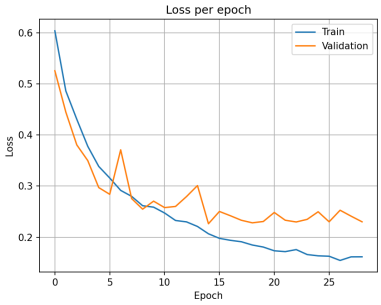
Parameter	Value
Optimizer	AdamW
Learning rate	10^{-4}
Weight decay	10^{-3}
Batch size	8
Gradient accumulation steps	2
Max epochs	100
Early stopping patience	14
Learning rate scheduler	ReduceLROnPlateau
Scheduler patience	4
Scheduler factor	0.5
Minimum LR	10^{-6}
Scheduler threshold	10^{-3}
Input sequence length	150,000 bytes
Padding token	256

A multi-crop evaluation strategy was applied: for each file, the model averaged predictions from one center crop and six random crops, reducing positional sensitivity and improving recall.

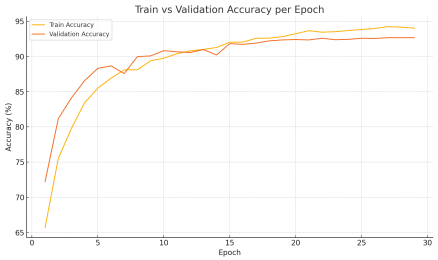
TABLE VI
EFFECT OF MULTI-CROP EVALUATION ON 1D CNN

Evaluation	Accuracy	Precision (0/1)	Recall (0/1)	F1-Score (0/1)	Spyware FN
Single Crop	91.96%	90.84% / 93.00%	92.31% / 91.65%	91.57% / 92.32%	167
Multi-Crop (7x)	**92.88%**	**92.86%** / 92.91%	92.03% / **93.65%**	**92.44%** / **93.28%**	**127**

Multi-crop evaluation improved overall accuracy to 92.88% and substantially reduced false negatives for the spyware class.



(a) Training and validation loss per epoch



(b) Training and validation accuracy per epoch

Fig. 3. Performance curves for the 1D CNN model: (a) loss and (b) accuracy per epoch.

VI. ENSEMBLE VOTING (COUNCIL SYSTEM)

The final system ensembles the six models described above. For each sample, the council outputs a confidence score of all judges. A sample is classified as spyware if at least half of the judges assign a confidence score exceeding 0.5.

TABLE VII
COUNCIL (ENSEMBLE) PERFORMANCE

Accuracy	Precision	Recall	F1-Score
97.00%	95.45% / 98.46%	98.33% / 95.80%	96.87% / 97.11%

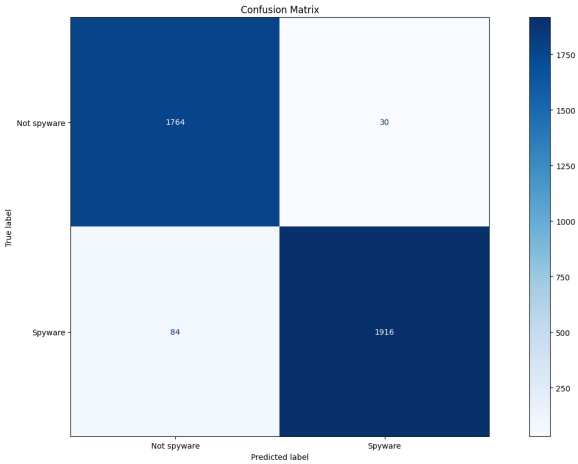


Fig. 4. Confusion Matrix for the Council Ensemble

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 models, 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

We 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.