Anusandhanvallari
Vol 2026, No.1
August 2026
ISSN 2229-3388

An Advanced Cybersecurity Framework for Malware Detection Using Byte N-Grams, Opcode Patterns, ASM Visualization, and XGBoost Classification

[1]Kushal Ram, [2]Bhartee Singh

[1]M. Tech Scholar, Department of Computer Science & Engineering, Compucom Institute of Technology & Management, Jaipur

[2]Assistant Professor, Department of Computer Science & Engineering, Compucom Institute of Technology & Management, Jaipur

Abstract: The fast growth in the emergence of malware has resulted in the identification of malware detection as one of the vital problems in current cybersecurity issues. Signature-based approaches to malware detection have been proved to have numerous shortcomings in detecting polymorphic and metamorphic forms of malware. This paper describes the evaluation of an advanced cybersecurity solution, which utilizes several static analysis approaches – namely Byte N-Gram features extraction, Opcode sequences analysis, and Assembly (ASM) language representation – along with the XGBoost gradient boosting model. This approach is benchmarked against several well-known baseline models including Random Forest, Support Vector Machine (SVM), Naïve Bayes, and deep learning frameworks. The experiments were conducted on the Microsoft Malware Classification Challenge (BIG 2015) dataset, which consists of 10,868 examples of malware from nine different families. According to the results obtained from the experiments, the integrated framework shows the best classification performance of 98.6% accuracy, 98.3% precision, 97.9% recall, and 98.1% F1-score.

Keywords: Malware Detection, Byte N-Grams, Opcode Patterns, ASM Visualization, XGBoost, Static Analysis, Feature Engineering, Cybersecurity, Machine Learning, Binary Classification

1. Introduction

The transition to digitalization in infrastructure all over the world has led to an explosive growth in attack surfaces for attackers. Malware including viruses, worms, trojans, ransomware, spyware, and rootkits remains one of the most resilient and constantly changing threats to cybersecurity of an organization. As reported by AV-Test Institute, more than 450,000 new malicious software samples are added to the database daily, illustrating the inefficiency of using manually crafted signatures.

Detection of malicious programs through the analysis of byte sequences or hash values, which is the leading method used by commercial anti-virus products, is computationally efficient but reactive. It fails to detect zero-day attacks and advanced forms of malware that use various methods such as code obfuscation, encryption, packing, or metamorphic engines to avoid detection. Behavioral analysis and dynamic analysis provide alternative views but add overhead to the process and can be overcome by environment-aware malware.

Machine learning (ML) based methods of malware detection have become quite popular as promising contenders in the field. Static analysis involves analysis of binary files without running them and yields abundant features based on file structure, byte patterns, disassembly information, and library imports. More importantly, static features are extracted before the file is executed, hence making it possible to detect malware without any hazards related to the dynamic analysis of files in production environment.

In this paper, we propose an approach for extraction and fusion of three different types of static feature modalities, namely: (1) Byte N-Grams, which involve byte patterns of files; (2) Opcode Frequency/Transition features, which

are extracted from disassembled assembly files; and (3) ASM Visualization features, which convert assembly files into grayscale images. These are then fed to an XGBoost classifier, which handles high dimensional and heterogeneous feature sets much better than others.

2. Related Work

2.1 Signature-Based Detection

Standard antivirus software depends on signature databases which are created and updated by the security firms. The first detailed examination of the weaknesses of signature based detection was performed by Szor (2005), especially with respect to polymorphic viruses. Although the method is nearly 100 percent accurate when dealing with known threats, it is highly inaccurate for new malware families.

2.2 Machine Learning Approaches

Machine learning for the purposes of malware detection was first introduced by Schultz et al. (2001), who showed that PE headers, imported DLL calls, and byte sequences can be used to train classifiers. Byte n-grams combined with the Boosted Decision Tree and Naive Bayes algorithms were studied by Kolter and Maloof (2006), who have shown that short byte sequences capture binary patterns, and that the length of n-grams between 4–6 bytes was most efficient.

SVM-based classification with dynamic analysis traces was presented by Rieck et al. (2008), while frequency-based static analysis features were tested by Santos et al. (2009). Static PE-based feature set for classification was used by Ravi and Manoharan (2012) in order to reach 97.3% classification accuracy in a balanced binary classification problem. Shabtai et al. (2012) suggested continuous monitoring with an online learning approach to cope with concept drift.

2.3 Deep Learning for Malware Detection

The visual approach used by Nataraj et al. (2011) involves converting malware binaries to gray scale images and using image processing techniques for classification purposes. Their algorithm, which uses GIST texture descriptors, showed that different families of malware are visually distinguishable in terms of the byte patterns especially in the entropy and structure of their code segments. The subsequent research by Gibert et al. (2019) built upon this visual classification approach using CNNs and obtained top results for visual malware classification benchmarks.

LSTM and GRU architectures have been explored using opcodes as sequences and the instruction streams after disassembly as natural language sequences. Athiwaratkun and Stokes (2017) showed that word2vec embeddings of opcodes along with LSTM classification performed competitively and provided interpretable intermediate representations.

2.4 Opcode-Based Analysis

The discriminative feature analysis based on frequency of opcode n-grams was proposed by Masud et al. (2008). Assembly mnemonics frequency distributions (for instance, MOV, PUSH, CALL, JMP) are different in various malware families due to the presence of certain algorithms in their packing, encryption procedures, and exploits. These distinctions were statistically analyzed by Bilar (2007).

2.5 Ensemble Methods and XGBoost

The Gradient Boosting Machines (Chen and Guestrin, 2016) have implemented XGBoost, which is a scalable tree-boosting system, and this has led to its success in structured data classification competitions. Some of the strengths of XGBoost include the regularization of parameters, dealing with sparse features, and customization of objective functions. Anderson & Roth (2018) used XGBoost on PE header features for malware detection,

showing better results compared to Random Forest and SVM baselines. Raff et al. (2018) proposed an approach for end-to-end byte level malware classification using deep learning.

3. Theoretical Background

3.1 Byte N-Grams

An N-gram can be described as the contiguous N elements from a particular sample. In case of the analysis of binary files, the byte N-grams are the contiguous N elements which have been obtained from the raw binary format of the executable file. Mathematically, in case of a binary file $B = \{b_1, b_2, ..., b_m\}$ of length m, the N-grams are:

$$\mathbf{NG(B, N) = \{(b_i, b_{i+1}, ..., b_{i+n-1}) \mid 1 \leq i \leq m - N + 1\}}$$

Each document's feature vector is defined as the distribution of frequencies among the vocabulary of observed N-grams. The number of possible bytes is 256^N (65,536 for N=2) in theory, however in practice, thresholding and dimensionality reduction are applied to overcome computational complexity. The N-gram representation is based on syntactical patterns at a very low level and does not need code disassembly and semantics of the code, which makes it resistant to code obfuscation.

3.2 Opcode Patterns

The opcodes in the assembly language form the mnemonic representation of the machine instructions obtained via disassembly of binary programs. Via software like IDA Pro and objdump, the executable is converted into a stream of assembly codes, which are then used to obtain the opcodes. The vector of opcode frequencies provides a concise way of representing the computational patterns in the program.

In addition to simple frequencies of individual opcodes, opcode N-grams represent the sequences of instructions that describe unique algorithmic operations such as encryption loops, stack manipulation procedures, and system call sequences.Opcode transition matrices may also be analyzed in terms of Markov chains.

3.3 ASM Visualization

The visualization method works by converting binary files/ASM files into gray scale images using the byte values of the file as pixel intensity values (0-255). The binary file is loaded as a one dimensional vector of bytes and transformed into a two dimensional matrix of constant width (usually 256/512 columns) depending on file size and height depending on file size. The resulting image will capture macrostructure patterns such as:

- Areas that appear dark representing null padded areas
- High entropy areas representing encryption/compression areas
- Texture patterns in the code areas
- Different header patterns in the boundary areas of the file

These textures can then be used to extract GIST descriptors, Local Binary Pattern (LBP), Histogram of Oriented Gradient (HOG) and CNN features from the image to classify the malware into different families using image similarity.

3.4 XGBoost Classification

The XGBoost (eXtreme Gradient Boosting) is an algorithm used for ensemble learning and uses gradient descent optimization to create a model of K weak learners (decision trees). The prediction function is:

$$\mathbf{\hat{y}_i = \Sigma_k f_k(x_i), f_k \in F}$$

where F is the space of regression trees. The objective function minimizes the regularized loss:

$$L = \Sigma_i\, l(y_i, \hat{y}_i) + \Sigma_k\, \Omega(f_k)$$

$\Omega(f) = \gamma T + \frac{1}{2}\lambda||w||^2$ where $\Omega(f)$ is the regularizer for tree complexity (in terms of number of leaves T and weight values at leaves). The main strengths of XGBoost for classifying malware features are as follows:

- Sparse feature matrix processing
- Second-order gradient descent for faster convergence
- Variance reduction through column subsampling
- Cross-validation support

4. Proposed Framework Architecture

4.1 System Overview

The proposed methodology is performed in a pipeline architecture using four main components: (1) Feature Extraction, where three pipelines process the binary files and the ASM files separately; (2) Feature Processing and Normalization, where each of the extracted feature sets is individually processed, filtered, and normalized; (3) Feature Fusion, where the three feature sets are combined to form a single high-dimensional feature vector; and (4) XGBoost Classification, where the high-dimensional feature vector is classified using the XGBoost model.

4.2 Byte N-Gram Extraction Pipeline

The byte n-gram pipeline proceeds as follows. Raw binary files are loaded into memory as byte arrays without prior preprocessing and parsing based on file formats (PE, ELF, Mach-O), ensuring flexibility of the algorithm to various executable file formats. Second, n-gram enumeration produces all n-grams for $n \in \{2, 3, 4\}$. Third, vocabulary construction is done by filtering out n-grams with frequency lower than a minimum threshold tf_min = 5 and retains only n-grams from the vocabulary V with $|V| \approx 100{,}000$. Fourth, the TF-IDF weighting step calculates weights of the terms to emphasize discriminative n-grams. Finally, Variance Thresholding and Chi-squared feature selection reduces the number of features down to the top 5,000.

Reasons for choosing several lengths of n-grams are as follows. Different lengths provide complementary information; thus, 2-grams capture two consecutive bytes, 3-grams are intermediate sequence of three bytes, and 4-grams represent longer sequence context. Fusion of the features from several n-gram lengths weighted by cross-validation performance produces a more informative representation.

4.3 Opcode Pattern Extraction Pipeline

The assembly files (.asm) are processed to obtain opcode sequences. The processing pipeline involves:

use of regular expressions for tokenization to detect instructions' mnemonics from the disassembled code, conversion of various instruction variants (e.g., MOV, MOVSX, MOVZX converted to the family of MOV instructions), and creation of the frequency vector of the top-500 most common opcodes. Moreover, 2-grams and 3-grams of opcode sequences are detected to account for instruction co-occurrence, and the transition matrices of opcodes are encoded as flattened probability distributions.

The dimension of the resulting opcode feature vector is ≈ 8,000, taking into account unigram frequency, bigram co-occurrence, and trigram patterns. The vector is L1-normalized prior to fusion.

4.4 ASM Visualization and Feature Extraction

The conversion process of ASM files into grayscale images involves the following steps: (1) reading the ASM file as a raw byte array; (2) converting the bytes to uint8 pixel values; (3) padding the one-dimensional array to

the closest square size and reshaping it into an image of fixed width (width = 256 pixels); (4) resizing the image to 256 × 256 with bilinear interpolation.

Visual features extracted include: GIST descriptors (512-dimensional perceptual scene descriptor capturing orientation and frequency content), Local Binary Pattern histograms (256-dimensional rotation-invariant texture descriptor), Global color histogram of pixel intensity distribution (64-dimensional), and image statistics including mean, variance, skewness, kurtosis, and entropy per image quadrant. The visual feature vector totals 900 dimensions.

4.5 Feature Fusion and XGBoost Configuration

Feature vectors from the three pipelines are concatenated: [F_ngram (5,000) || F_opcode (8,000) || F_visual (900)] → F_fused (13,900 dimensions). The fused vector is standardized using Z-score normalization (μ=0, σ=1) per feature across the training set.

The XGBoost classifier is configured with the following hyperparameters, optimized via Bayesian hyperparameter search with 5-fold cross-validation: n_estimators = 500, max_depth = 8, learning_rate = 0.05, subsample = 0.8, colsample_bytree = 0.7, gamma = 0.1, reg_lambda = 1.5, reg_alpha = 0.1, and objective = 'multi:softprob' for multi-class probability output. Early stopping with patience = 50 rounds prevents overfitting.

5. Experimental Setup

5.1 Dataset Description

The experiments are performed using the data from the Microsoft Malware Classification Challenge (BIG 2015) competition, which has been made publicly available by Microsoft Research and Kaggle. The dataset consists of 10,868 samples of malware belonging to 9 different malware families. The malware samples are presented in two formats – as raw binary files (.bytes, with hex dump of binary code) and disassembled assembly (.asm).

Table 1: Malware Family Distribution in BIG 2015 Dataset

Malware Family	Common Name	Samples	Category
Class 1	Ramnit	1,541	Worm
Class 2	Lollipop	2,478	Adware
Class 3	Kelihos_ver3	2,942	Backdoor
Class 4	Vundo	475	Trojan
Class 5	Simda	42	Backdoor
Class 6	Tracur	751	Trojan Downloader
Class 7	Kelihos_ver1	398	Backdoor
Class 8	Obfuscator.ACY	1,228	Obfuscated
Class 9	Gatak	1,013	Trojan

Malware Family	Common Name	Samples	Category
Total	—	10,868	9 Families

5.2 Baseline Models

In order to set comparative benchmarks, the following classifiers are being assessed based on the common feature set:

- Naïve Bayes (NB): Multinomial NB with Laplace smoothing (α=1.0). It is frequently used as the benchmark classifier in the context of text classification tasks.
- Support Vector Machine (SVM): Linear kernel with C=1.0. Evaluated based on the TF-IDF byte n-grams due to limitations of the kernel matrix calculations.
- Random Forest (RF): 500 trees, max_depth=None, min_samples_split=2, full feature fusion.
- Multi-layer Perceptron (MLP): 3 layers (1024, 512, 256 neurons), activation='relu', dropout=0.3, Adam optimizer, 100 epochs training.
- Convolutional Neural Network (CNN): Applied only to the ASM visualization images. 4 Conv2D blocks (32→64→128→256), GlobalAverage

5.3 Evaluation Protocol

Data is divided into a 80% training (8,694 records) and a 20% testing (2,174 records) set using stratified sampling. All hyperparameters are optimized on the validation set, which is a 20% sample of the training data. The final performance metrics are obtained on the test set and are: Accuracy, Precision (macro-average), Recall (macro-average), F1 Score (macro-average), Area Under the ROC Curve (AUC-ROC, One-vs-Rest), and Matthews Correlation Coefficient (MCC).

6. Results and Analysis

6.1 Classification Performance Comparison

In Table 2, there is the detailed performance comparison of all evaluated models. The proposed framework of XGBoost using fused features shows superior results compared to all baselines on all criteria.

Table 2: Performance Comparison of Classification Models

Model	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)	AUC-ROC	MCC
Naïve Bayes	82.4	81.7	80.3	81.0	0.934	0.801
SVM (Linear)	91.2	90.8	90.1	90.4	0.971	0.898
Random Forest	95.8	95.3	94.9	95.1	0.989	0.952

Model	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)	AUC-ROC	MCC
MLP	96.3	95.9	95.6	95.7	0.991	0.958
CNN (Visual)	94.7	94.2	93.8	94.0	0.987	0.941
XGBoost	98.6	98.3	97.9	98.1	0.998	0.984

6.2 Per-Class Performance Analysis

The analysis of class-level evaluation metrics shows that our proposed approach performs the best for the classes of Kelihos_ver3 (F1 = 99.4%) and Lollipop (F1 = 99.2%), as both of these classes have a larger number of samples with unique byte-level features. The hardest class is Simda (F1 = 94.8%) due to its small sample size (n = 42), leading to serious class imbalance.

6.3 Contribution of Feature Modalities

To quantify the contribution of each feature modality, we conduct an ablation study training XGBoost with individual and combined feature sets.

Table 3: Feature Modality Contributions

Feature Configuration	Accuracy (%)	F1-Score (%)	AUC-ROC	Features
Byte N-Grams Only	94.2	93.8	0.982	5,000
Opcode Patterns Only	92.7	92.1	0.978	8,000
ASM Visualization Only	91.5	91.0	0.975	900
N-Grams + Opcode	96.8	96.4	0.993	13,000
N-Grams + Visual	96.1	95.7	0.991	5,900
Opcode + Visual	95.3	94.9	0.989	8,900
All Three (Proposed)	98.6	98.1	0.998	13,900

The results confirm a clear and consistent synergistic effect of feature fusion: the combined three-modality system (98.6%) substantially outperforms the best single-modality system (Byte N-Grams: 94.2%) by 4.4 percentage points. Pairwise combinations also demonstrate improvements over individual modalities, with N-Grams + Opcode achieving 96.8%, indicating that byte-level and instruction-level representations are particularly complementary.

6.4 Feature Importance Analysis

XGBoost's built-in feature importance scores (based on gain—the average improvement in split purity contributed by a feature across all trees) reveal the relative discriminative power of features. The top-20 most important features comprise: 8 byte N-gram features (predominantly 4-gram patterns at specific offsets in PE headers), 9 opcode features (notably PUSH, MOV, and CALL instruction frequencies and bigrams), and 3 visual features (image entropy statistics and LBP histogram bins).

Such a distribution is indicative of the fact that despite the fact that the visual features make the contribution towards discriminative power of the ensemble, individual predictive power of the byte and instruction-level features is relatively greater than that of the former – in accordance with the results of the ablation experiment.

6.5 Computational Performance

Table 4 shows the training time, prediction time per data point, and memory used by each model.

Table 4: Computational Efficiency Comparison

Model	Training Time (min)	Inference (ms/sample)	Memory (GB)	GPU Required
Naïve Bayes	0.3	0.2	0.5	No
SVM	12.4	1.8	2.1	No
Random Forest	8.7	3.2	4.3	No
MLP	45.2	2.1	6.8	Yes
CNN	127.6	8.4	9.2	Yes
XGBoost (Ours)	18.3	4.7	5.6	No

XGBoost attains best-in-class accuracy (98.6%) in 18.3 minutes of training time, which is considerably faster compared to deep learning methods (45.2–127.6 minutes) without using any GPU setup. This feature is essential for implementing it in environments with limited security operational resources.

7. Discussion

The high efficiency of the suggested framework is due to the complementing nature of the three feature types. Byte n-gram features account for the binary structure of the files which is not sensitive to the quality of the disassembling as some malware uses techniques that make disassembling impossible. The algorithmic structure captured by opcode features is less dependent on byte-level changes introduced by packers. The visualization of the ASM code allows to understand the structure of the files in general. The better performance of the XGBoost classifier in comparison with the Random Forest classifier is due to the fact that while being ensemble tree methods both use gradient boosting to improve accuracy, XGBoost uses gradient descent to optimize residuals that cannot be solved using Random Forest as it builds trees in parallel and independently.

A major challenge in detecting malware is that of adversarial evasion: advanced malware creators can intentionally design samples to evade detectors. Multi-modal properties of our model provide some inherent robustness

characteristics. It is necessary for an attacker to evade detection using multiple modalities at once—three, in this case: byte level, instruction level, and visual structural—and it poses much more engineering challenges than evading a single modality. Byte level obfuscation techniques (for example, adding NOP bytes, dead code) that impact n-gram feature generation will modify the frequency distribution of opcodes, thus identifying unusual instruction sequences. However, code transposition attacks on opcode n-gram features have byte level footprints detectable by the n-gram model.

8. Conclusion

This paper has introduced a complete comparative study of the sophisticated multi-class malware detection system utilizing Byte N-Gram feature extraction, Opcode pattern identification, ASM visualization and XGBoost classification. Experimental results on the popular BIG 2015 benchmark dataset containing 10,868 instances from nine different malware families have shown that the proposed multi-modal technique can achieve 98.6% accuracy and 98.1% macro-F1 score, significantly surpassing five existing baseline systems, including those based on deep learning. It was proved that each of the feature modalities has a significant impact on performance through ablation study, whereby byte n-grams are shown to be the most influential individual feature, and feature fusion demonstrates consistent synergy. The computational analysis has shown that the proposed technique can be efficiently deployed without requiring GPU resources, as it exhibits higher accuracy with comparable inference speed. The feature importance analysis is a valuable functionality which enables interpretability capabilities that can prove important in many cases. This paper proves that the proposed multi-modal multi-class malware detection system has high effectiveness, efficiency, and interpretability. Future research in resilient malware detection techniques should be based on the combination of the proposed multi-modal feature fusion and XGBoost classifier.

References:

[1] Anderson, H. S., & Roth, P. (2018). EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models. arXiv:1804.04637.

[2] Athiwaratkun, B., & Stokes, J. W. (2017). Malware Detection by Eating a Whole EXE. AAAI Workshop on Artificial Intelligence for Cyber Security.

[3] Bilar, D. (2007). Opcodes as Predictor for Malware. International Journal of Electronic Security and Digital Forensics, 1(2), 156–168.

[4] Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 785–794.

[5] Comar, P. M., Liu, L., Saha, S., Tan, P. N., & Nucci, A. (2013). Combining Supervised and Unsupervised Learning for Zero-Day Malware Detection. IEEE INFOCOM, 2022–2030.

[6] Gibert, D., Mateu, C., Planes, J., & Vicens, R. (2019). Using Convolutional Neural Networks for Classification of Malware Represented as Images. Journal of Computer Virology and Hacking Techniques, 15(1), 15–28.

[7] Kolter, J. Z., & Maloof, M. A. (2006). Learning to Detect and Classify Malicious Executables in the Wild. Journal of Machine Learning Research, 7, 2721–2744.

[8] Masud, M. M., Khan, L., & Thuraisingham, B. M. (2008). A Scalable Multi-Level Feature Extraction Technique to Detect Malicious Executables. Information Systems Frontiers, 10(1), 33–45.

[9] Microsoft Corporation. (2015). Microsoft Malware Classification Challenge (BIG 2015). Kaggle Competition Dataset. https://www.kaggle.com/c/malware-classification

[10] Nataraj, L., Karthikeyan, S., Jacob, G., & Manjunath, B. S. (2011). Malware Images: Visualization and Automatic Classification. Proceedings of the 8th International Symposium on Visualization for Cyber Security (VizSec 2011), 4:1–4:7.

[11] Raff, E., Barker, J., Sylvester, J., Brandon, R., Catanzaro, B., & Nicholas, C. K. (2018). Malware Detection by Eating a Whole EXE. Proceedings of the Workshop on Artificial Intelligence for Cyber Security, 268–286.

[12] Ravi, C., & Manoharan, R. (2012). Malware Detection Using Windows API Sequence and Machine Learning. International Journal of Computer Applications, 43(17), 12–16.

[13] Rieck, K., Holz, T., Willems, C., Düssel, P., & Laskov, P. (2008). Learning and Classification of Malware Behavior. Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA), 108–125.

[14] Santos, I., Brezo, F., Nieves, J., Penya, Y. K., Sanz, B., Laorden, C., & Bringas, P. G. (2009). Idea: Opcode-Sequence-Based Malware Detection. Engineering Secure Software and Systems (ESSoS), 35–43.

[15] Schultz, M. G., Eskin, E., Zadok, F., & Stolfo, S. J. (2001). Data Mining Methods for Detection of New Malicious Executables. Proceedings of the IEEE Symposium on Security and Privacy, 38–49.

[16] Shabtai, A., Moskovitch, R., Feher, C., Dolev, S., & Elovici, Y. (2012). Detecting Unknown Malicious Code by Applying Classification Techniques on OpCode Patterns. Security Informatics, 1(1), 1–22.

[17] Szor, P. (2005). The Art of Computer Virus Research and Defense. Addison-Wesley Professional, Upper Saddle River, NJ.

[18] Ye, Y., Li, T., Adjeroh, D., & Iyengar, S. S. (2017). A Survey on Malware Detection Using Data Mining Techniques. ACM Computing Surveys, 50(3), 41:1–41:40.