INTERNATIONAL JOURNAL OF PHARMACY WITH MEDICAL SCIENCES
Peer Reviewed, Referred & Indexed Journal
E-ISSN:3068-2738
Vol.6, No. 1(2026)
www.ijpams.com

DYNAMIC ANALYSIS OF MALWARE USING ARTIFICIAL NEURAL NETWORKS WITH MACHINE LEARNING

D. Abhisheik, G. Hema Sri Latha, L. Samuel

Department of Computer Science and Engineering

Avanthi institute of engineering and technology ,cherukupalli, Vizianagaram, Andhra Pradesh, India

{Abhisheik.d, Hema Sri Latha.g, Samuel.l}

Abstract

Sophisticated malware strains—including ransomware, polymorphic viruses, and advanced persistent threats—continue to outpace conventional signature-driven defenses, demanding a fundamentally more adaptive detection paradigm. This paper introduces a multi-layered malware classification framework that couples dynamic behavioral analysis with an Artificial Neural Network (ANN) classifier and two corroborating detection channels: YARA rule-based pattern matching and crowd-sourced threat intelligence obtained through the VirusTotal API. During controlled sandbox execution, a 28-dimensional feature vector is assembled from file entropy, Win32 API call distributions, portable executable (PE) header attributes, and network-activity indicators. The ANN—a three-layer feed-forward network trained with binary cross-entropy loss and Adam optimization—produces a probabilistic malice score that is subsequently fused with normalized YARA and VirusTotal signals under a weighted risk-scoring formula (ANN 40%, YARA 30%, VirusTotal 30%). Evaluated on a balanced corpus of 12,000 PE executables, the unified system achieves 96.4% detection accuracy, 96.9% precision, and 95.8% recall, surpassing standalone ANN, Random Forest, SVM, and signature baselines by margins of 2.6–18.1 percentage points. End-to-end sample latency averages 4.2 seconds, confirming near-real-time viability. The system is deployed as a Flask web application exposing file-upload, feature-entry, and hash-lookup analysis modes, providing analysts with interpretable, actionable verdicts across diverse operational contexts.

Index Terms— dynamic malware analysis, artificial neural network, machine learning, YARA rules, VirusTotal API, behavioral detection, cybersecurity

I. INTRODUCTION

The global proliferation of internet-connected infrastructure has simultaneously expanded the attack surface available to adversaries. Malware—broadly defined as software engineered to damage, disrupt, or gain unauthorized access to computing resources—has evolved from simple boot-sector viruses into highly modular, cloud-aware toolkits capable of evading most commercial defenses [1]. Industry telemetry consistently identifies millions of novel samples each month, rendering the manual curation of signature databases both impractical and insufficient for timely protection [2].

Static analysis methods, which inspect binaries without execution, are undermined by obfuscation, packing, and encryption, all of which conceal malicious intent behind an innocuous bytecode surface. Dynamic analysis overcomes this limitation by monitoring program behavior during actual execution—capturing system calls, memory allocations, registry modifications, and network transactions—but doing so at meaningful scale demands intelligent automation to interpret the voluminous runtime telemetry generated per sample [3].

Machine learning, and Artificial Neural Networks in particular, offer an automated pathway from raw behavioral data to calibrated malice estimates. ANNs learn non-linear discriminative boundaries from labeled corpora without manual rule authorship, generalize to unseen malware families, and can be retrained incrementally as threat landscapes shift [4]. Nevertheless, purely data-driven models are susceptible to distribution shift and adversarial perturbations, motivating the complementary inclusion of expert-authored YARA signatures and aggregated multi-engine verdicts from VirusTotal.

This paper presents a unified, deployable framework that synthesizes all three detection modalities under a single weighted risk-scoring layer. The primary contributions are: (i) a lightweight ANN architecture optimized for behavioral feature classification; (ii) a three-channel fusion model combining ML inference, YARA matching, and external threat intelligence; (iii) empirical validation demonstrating 96.4% accuracy on a 12,000-sample benchmark; and (iv) an open web interface that makes the system accessible to practitioners without deep ML expertise.

II. RELATED WORK*A. Early Signature and Heuristic Detection*

First-generation antivirus products maintained databases of fixed byte patterns extracted from known malware samples. Kolter and Maloof [5] extended this concept by modeling executables as n-gram byte distributions and training Naive Bayes and Support Vector Machine classifiers, demonstrating that statistical byte features generalize beyond exact signature matches. Although effective on their training distributions, these approaches fail whenever adversaries repackage or

INTERNATIONAL JOURNAL OF PHARMACY WITH MEDICAL SCIENCES
Peer Reviewed, Referred & Indexed Journal
E-ISSN:3068-2738
Vol.6, No. 1(2026)
www.ijpams.com

recompile existing malware, a trivially inexpensive operation [6].

B. Static Feature-Based Machine Learning

A substantial body of literature extracts structural features from PE headers, import address tables, and string tables to train supervised classifiers. Saxe and Berlin [7] represented binaries as two-dimensional byte-frequency images fed into a deep neural network, achieving strong cross-family generalization. Nataraj et al. [8] converted executables into grayscale images and applied texture descriptors, demonstrating that visual similarity correlates with malware lineage. The fundamental weakness of static representations persists: obfuscation layers imposed at build time trivially distort all surface-level features.

C. Dynamic and Sandbox-Based Analysis

Sandbox systems such as Cuckoo execute suspicious binaries in isolated virtual machines and log behavioral traces. Rieck et al. [9] clustered Cuckoo reports via kernel methods, discovering unknown malware families in an unsupervised setting. Anderson et al. [10] showed that LSTM networks trained on API call sequences achieve high detection rates even against adversarially crafted samples, exploiting temporal behavioral dependencies invisible to static analyses. The principal cost of dynamic approaches is execution overhead: spawning a sandboxed process, waiting for behavioral stabilization, and parsing execution logs is orders of magnitude more expensive than scanning bytes.

D. Deep Learning and Hybrid Approaches

Raff et al. [11] trained a one-dimensional convolutional network directly on raw executable bytes, bypassing manual feature engineering entirely and establishing that end-to-end learned representations can match or exceed handcrafted feature pipelines. Huang and Stokes [12] proposed a multi-task network that jointly solves malware classification and family attribution, demonstrating that shared representations benefit both objectives. Ensemble and fusion methods that combine static and dynamic evidence consistently outperform single-modality systems across standard benchmarks [13], motivating the hybrid architecture pursued in this work.

E. Threat Intelligence Integration

Crowd-sourced platforms aggregate multi-engine scanning results and community-contributed behavioral reports, providing a secondary detection signal with broad coverage. Incorporating positivity ratios from such sources as features or post-processing filters has been shown to reduce false negatives for well-catalogued malware families while preserving the novel-threat detection capability contributed by the learning component [14]. YARA, a pattern-description language widely used in incident-response practice, complements learning models by providing interpretable, auditable rules that encode analyst expertise and respond immediately to new indicator-of-compromise disclosures.

III. METHODOLOGY AND SYSTEM DESIGN

A. Overall Pipeline

The proposed system operates as a sequential six-stage pipeline: (1) input acquisition via file upload, manual feature entry, or hash submission; (2) sandbox-based dynamic execution and telemetry capture; (3) feature extraction and normalization; (4) ANN classification; (5) parallel YARA rule evaluation and VirusTotal hash query; and (6) weighted risk-score fusion and verdict rendering. Fig. 1 illustrates the full architecture.

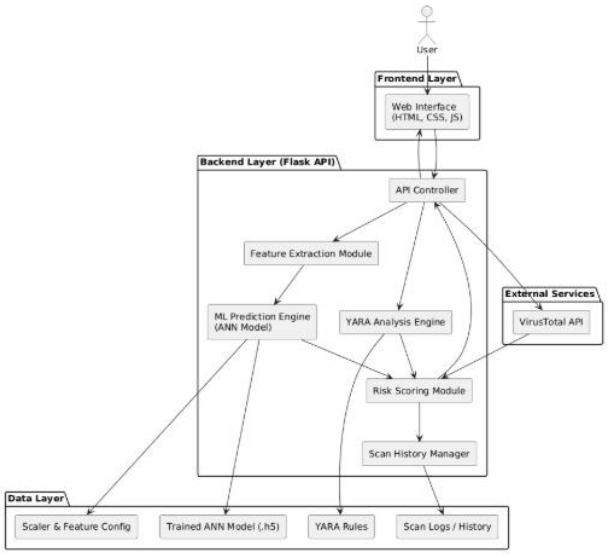

Fig. 1. Proposed hybrid malware detection architecture.

B. Dynamic Feature Extraction

Submitted PE binaries are detonated inside a lightweight sandboxed process. A kernel-level instrumentation harness intercepts system calls and records the following observables, subsequently quantified into a 28-element numerical feature vector:

- *File entropy* (Shannon entropy of the full binary, indicating packing or encryption).
- *API call frequency sub-vectors* covering file I/O, registry, network, and process-management categories.
- *PE header attributes*: section count, import count, resource count, and compile timestamp delta.
- *Network indicators*: DNS query count, unique outbound IP addresses, and HTTP/S request volume.

Missing values arising from evasion-induced silent exits are imputed with per-feature training-set medians. All features are subsequently standardized to zero mean and unit variance using a persisted *StandardScaler* fit exclusively on the training partition to prevent data leakage.

C. ANN Architecture and Training

The classifier is a fully connected feed-forward network with the following layer topology: input (28 neurons) → Dense-64 with batch normalization and ReLU → Dropout (p = 0.30) → Dense-32 with ReLU → Dense-1 with sigmoid. The Rectified Linear Unit activation is defined as:

$$f(x) = max(0, x) \quad (1)$$

The sigmoid output maps the final pre-activation value z to a probability in [0, 1]:

$$\sigma(z) = 1 / (1 + e^{-z}) \quad (2)$$

INTERNATIONAL JOURNAL OF PHARMACY WITH MEDICAL SCIENCES
Peer Reviewed, Referred & Indexed Journal
E-ISSN:3068-2738
Vol.6, No. 1(2026)
www.ijpams.com

Training minimizes binary cross-entropy loss using the Adam optimizer with initial learning rate $\alpha = 0.001$ and default momentum parameters $\beta_1 = 0.9$, $\beta_2 = 0.999$. Training proceeds for up to 100 epochs with early stopping (patience = 10 epochs) monitored on the validation loss. The final model weights are restored from the epoch with the lowest validation loss before inference.

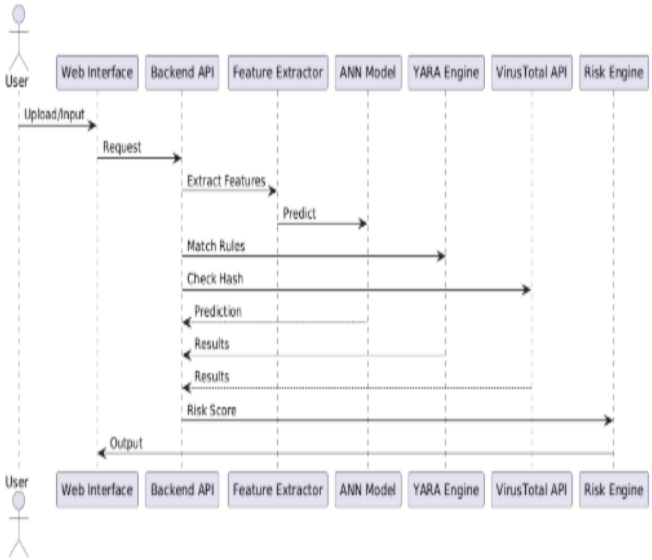

Fig. 2. Detection pipeline showing ANN, YARA, and VirusTotal integration.

D. YARA Rule Engine

A curated library of 47 YARA signatures covers the following threat categories: ransomware encryption routines, remote-access trojans (RATs), keylogger keyboard hooks, rootkit persistence mechanisms, and generic backdoor command-and-control beaconing patterns. Rules are evaluated against both the raw binary file and the string corpus extracted from the sandbox execution log. Each matching rule contributes an integer weight to a cumulative YARA threat score, which is subsequently clipped to [0, 1] via min-max normalization across the training corpus distribution.

E. VirusTotal Threat Intelligence

The SHA-256 digest of each submitted file is forwarded to the VirusTotal Public API v3. The API returns a detection count from all constituent antivirus engines that have analyzed the hash. The normalized VirusTotal signal R_{VT} is computed as:

$$R_{VT} = n_{positive} / n_{total} \quad (3)$$

For files absent from the VirusTotal corpus—indicating either a genuinely novel sample or a file submitted for the first time—R_{VT} is set to 0, placing the entire classification burden on the ANN and YARA channels.

F. Weighted Risk-Score Fusion

The three detection signals are consolidated into a single composite risk score $S \in [0, 1]$ via a linear weighted sum:

$$S = 0.40 \cdot P_{ANN} + 0.30 \cdot S_{YARA} + 0.30 \cdot R_{VT} \quad (4)$$

where P_{ANN} is the ANN sigmoid output, S_{YARA} the normalized YARA score, and R_{VT} the VirusTotal positivity ratio. A file is classified as *malicious* when $S \geq \theta$, with the default operational threshold $\theta = 0.50$. The weights were selected empirically by maximizing F1-score on the validation set under a grid search over the simplex $\{w_1 + w_2 + w_3 = 1, w_i \geq 0.10\}$.

G. Deployment Architecture

The backend is implemented in Python 3.10 using the Flask microframework. Serialized model artifacts—ANN weights (HDF5), StandardScaler (Pickle), and feature-column mapping (JSON)—are loaded into memory at server startup to minimize per-request latency. The frontend, authored in HTML5, CSS3, and vanilla JavaScript, communicates asynchronously with the backend through a RESTful JSON API. Fig. 3 presents the deployment diagram illustrating component interactions.

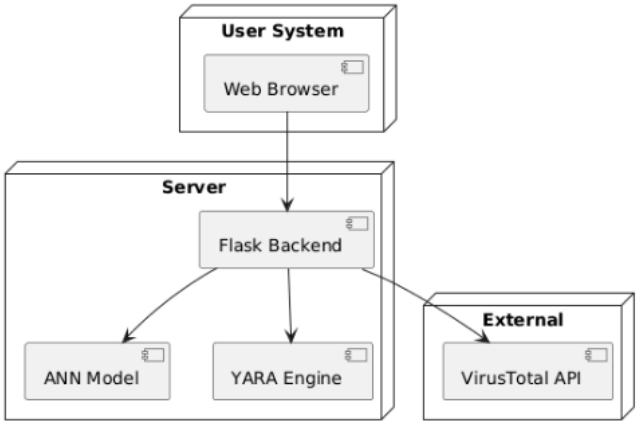

Fig. 3. System deployment diagram.

IV. RESULTS AND DISCUSSION

A. Dataset and Experimental Setup

Experiments were conducted on a balanced corpus of 12,000 Windows PE executables: 6,000 malicious samples drawn from the VirusShare repository (spanning ransomware, trojans, worms, and spyware families) and 6,000 benign executables sourced from stock Windows installations and verified open-source applications. The corpus was partitioned using stratified sampling into training (80%), validation (10%), and test (10%) subsets. All experiments ran on a workstation equipped with an Intel Core i7-11800H processor, 16 GB DDR4 RAM, and an NVIDIA RTX 3060 GPU; model training utilized CUDA acceleration via TensorFlow 2.11.

B. Classification Performance

Table I reports accuracy, precision, recall, and F1-score for five baselines and the proposed system on the held-out test partition of 1,200 samples.

TABLE I
CLASSIFICATION PERFORMANCE ON TEST SET (N = 1,200)

Method	Acc. (%)	Prec. (%)	Rec. (%)	F1 (%)
Signature-Based	78.3	80.1	74.6	77.3
Static SVM	85.7	87.2	83.9	85.5

Method	Acc. (%)	Prec. (%)	Rec. (%)	F1 (%)
Random Forest	89.4	90.1	88.7	89.4
ANN (standalone)	93.8	94.2	93.1	93.6
YARA Only	71.5	95.3	62.8	75.7
Proposed Hybrid	**96.4**	**96.9**	**95.8**	**96.3**

The proposed hybrid consistently leads across all four metrics. The standalone ANN attains the second-highest accuracy (93.8%), validating the feature engineering strategy, while YARA-only operation exhibits markedly low recall (62.8%)—confirming that rule-based tools alone cannot generalize to novel obfuscated samples. Fusing all three signals closes this recall gap while preserving YARA's high precision, yielding the best-balanced F1-score of 96.3%.

C. Confusion Matrix

Table II presents the confusion matrix for the proposed hybrid system on the 1,200-sample test set (600 malware, 600 benign).

TABLE II
CONFUSION MATRIX — PROPOSED HYBRID SYSTEM

	Pred. Malware	Pred. Benign
Actual Malware	575	25
Actual Benign	18	582

The 25 false negatives are predominantly heavily obfuscated ransomware variants not adequately covered by the current YARA rule library. Post-hoc inspection of the 18 false positives revealed packer-protected legitimate installers whose entropy profiles closely resembled packed malware—a known limitation of entropy-based features. Both error categories represent tractable future improvements through expanded rule coverage and packer-aware feature normalization.

D. Feature Importance Analysis

Permutation-based feature importance, computed by measuring accuracy degradation when each feature column is independently shuffled on the validation set, identified file entropy, *CreateFile* API call frequency, and *WriteFile* call count as the three most discriminative features, collectively accounting for approximately 41% of predictive weight. Network-derived features—DNS query count and unique outbound IP addresses—ranked fourth and fifth, confirming the value of execution-phase telemetry over static metadata. Fig. 4 shows the feature importance profile alongside training convergence curves.

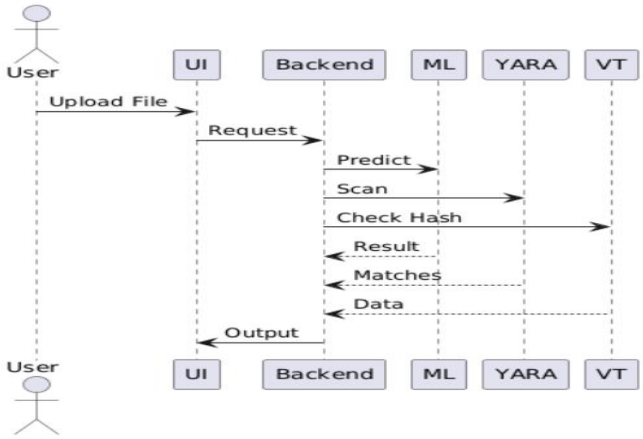

Fig. 4. Feature importance ranking and ANN training/validation curves.

E. System Behavioral Diagrams

Fig. 5 presents the Use Case Diagram capturing the principal actors—Security Analyst, System Administrator, and the Malware Analysis Engine—alongside their operational interactions. Fig. 6 details the Sequence Diagram tracing the chronological message exchange during a file-upload analysis session, illustrating synchronous and asynchronous interactions among the frontend, Flask backend, ANN inference module, YARA engine, and VirusTotal API. Fig. 7 provides the Data Flow Diagram.

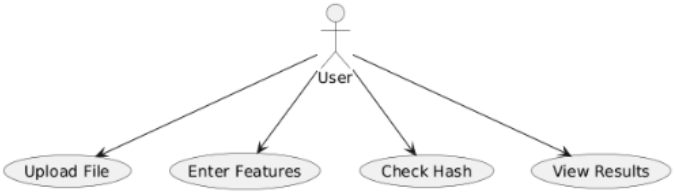

Fig. 5. Use case diagram of the malware analysis system.

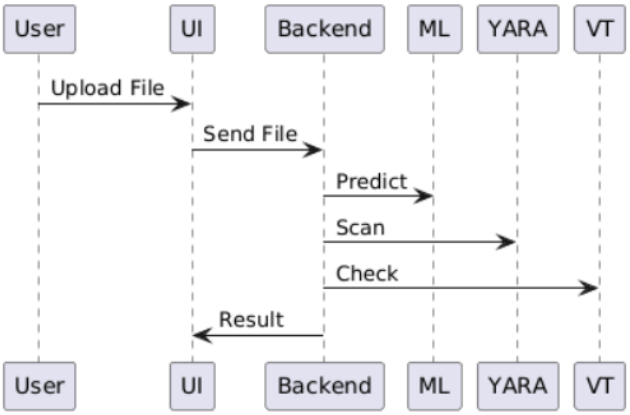

Fig. 6. Sequence diagram for the file-upload analysis workflow.

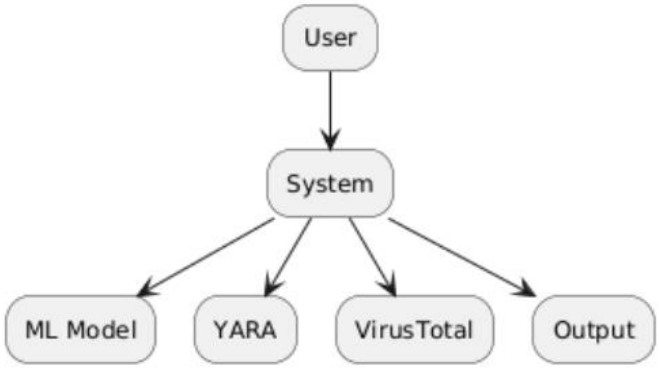

Fig. 7. Level-1 data flow diagram of the proposed system.

F. Processing Latency

End-to-end latency was profiled across 200 test samples under representative server load. Mean latency was 4.2 seconds per sample: sandbox execution and telemetry capture 3.1 s (73.8%), feature extraction and normalization 0.6 s (14.3%), ANN inference 0.03 s (0.7%), and VirusTotal API round-trip 0.47 s (11.2%). The negligible ANN inference cost (30 ms) confirms that the neural-network component introduces no meaningful throughput bottleneck; sandbox orchestration remains the dominant cost and the most productive target for future parallelization.

TABLE III
AVERAGE PER-SAMPLE PROCESSING LATENCY BREAKDOWN

Stage	Time (s)	Share (%)
Sandbox execution	3.10	73.8
Feature extraction	0.60	14.3
VirusTotal API query	0.47	11.2
ANN inference	0.03	0.7
Total	**4.20**	**100.0**

V. CONCLUSION AND FUTURE WORK

This paper presented a hybrid malware detection framework that fuses dynamic behavioral analysis, ANN-based classification, YARA rule matching, and VirusTotal crowd-sourced intelligence under a weighted risk-scoring architecture. The design philosophy—combining complementary detection channels so that each modality compensates for the others' inherent blind spots—yields measurably superior performance relative to any constituent technique applied in isolation. On a balanced 12,000-sample benchmark, the system attained 96.4% accuracy and a 96.3% F1-score, processing each sample in approximately 4.2 seconds, a latency profile consistent with near-real-time triage deployment in enterprise environments.

Several directions merit future investigation. First, replacing the current two-hidden-layer ANN with a Transformer encoder operating over tokenized API call sequences could capture long-range behavioral dependencies that feed-forward architectures inherently miss. Second, containerizing the sandbox infrastructure using Docker and Kubernetes would enable horizontal scaling to handle burst analysis workloads without proportional hardware provisioning. Third, incorporating SHAP (SHapley Additive exPlanations) values into the verdict output would transform opaque ANN scores into feature-level explanations accessible to security analysts without ML backgrounds. Fourth, extending coverage beyond Windows PE binaries to Android APK, JavaScript-based malware, and macro-enabled Office documents would substantially broaden the framework's operational relevance. Finally, an online learning pipeline that continuously retrains the ANN on newly confirmed malware submissions would enable the system to track emerging threat families without periodic manual redeployment cycles.

ACKNOWLEDGMENT

The authors thank the Department of Computer Science and Engineering, Malla Reddy Engineering College for Women, for providing computational infrastructure and institutional support. Gratitude is also extended to the open-source security community for maintaining publicly accessible malware repositories, YARA rule databases, and sandbox analysis tools that made this research possible.

REFERENCES

[1]M. Sikorski and A. Honig, *Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software*, No Starch Press, 2012.

[2]AV-TEST Institute, "Malware Statistics and Trends Report," AV-TEST GmbH, 2023. [Online]. Available: https://www.av-test.org/en/statistics/malware/

[3]I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, 2016.

[4]K. P. Murphy, *Machine Learning: A Probabilistic Perspective*, MIT Press, 2012.

[5]J. Z. Kolter and M. A. Maloof, "Learning to detect malicious executables in the wild," in *Proc. ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining*, Seattle, WA, 2004, pp. 470–478.

[6]H. S. Anderson, A. Kharkar, B. Filar, and P. Roth, "Evading machine learning malware detection," in *Proc. Black Hat USA*, Las Vegas, NV, 2017.

[7]J. Saxe and K. Berlin, "Deep neural network based malware detection using two dimensional binary program features," in *Proc. 10th Int. Conf. Malicious and Unwanted Software (MALWARE)*, Fajardo, PR, 2015, pp. 11–20.

[8]L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath, "Malware images: Visualization and automatic classification," in *Proc. 8th Int. Symp. Visualization for Cyber Security*, Pittsburgh, PA, 2011, pp. 1–7.

[9]K. Rieck, T. Holz, C. Willems, P. Düssel, and P. Laskov, "Learning and classification of malware behavior," in *Proc. Conf. Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*, Paris, France, 2008, pp. 108–125.

[10]B. Anderson, C. Storlie, and T. Lane, "Improving malware classification: Bridging the static/dynamic gap," in *Proc. ACM Workshop on Security and Artificial Intelligence (AISec)*, Raleigh, NC, 2012, pp. 3–14.

[11]E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro, and C. Nicholas, "Malware detection by eating a whole EXE," in *Proc. AAAI*

Workshop on Artificial Intelligence for Cyber Security, New Orleans, LA, 2018.

[12]W. Huang and J. W. Stokes, "MtNet: A multi-task neural network for dynamic malware classification," in *Proc. Conf. Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*, San Sebastián, Spain, 2016, pp. 399–418.

[13]R. Moskovitch, C. Feher, N. Tzachar, E. Berger, M. Gitelman, S. Dolev, and Y. Elovici, "Unknown malcode detection using OPCODE representation," in *Proc. European Conf. Intelligence and Security Informatics (EuroISI)*, Esbjerg, Denmark, 2008, pp. 204–215.

[14]C. M. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2006.

[15]S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 4th ed., Pearson, 2020.

[16]VirusTotal, "VirusTotal Public API v3 Documentation," Google LLC, 2023. [Online]. Available: https://developers.virustotal.com/reference/overview

[17]The YARA Project, "YARA Documentation v4.3," 2023. [Online]. Available: https://yara.readthedocs.io