



Multi-Layered Cyber Defense: Combining AI-Based Malware Classification, Server Monitoring, and Penetration Testing

Rupesh Khadka ^{*a}, Prajwal Rai^b, and Bibek Gautam^c

^aPadmashree College, Nilai University.

^bKantipur City College, Purbanchal University.

^cPulchowk Campus Tribhuvan University.

Abstract A multi-layered cybersecurity framework is suggested to tackle the growing complexity of cyber threats that frequently bypass traditional signature based antivirus solutions. The system combines three main elements: AI-based malware classification, live server monitoring, and penetration testing, all developed with the Flask framework and presented via a user friendly web interface. Highlighting flexibility and quick responsiveness, the system utilizes two machine learning algorithms Extra Trees Classifier and Logistic Regression for malware detection and classification, with the Extra Trees model reaching an outstanding accuracy of 99.24%. This elevated precision highlights its ability to detect both familiar and new threats. The server monitoring component identifies irregularities like unauthorized access attempts and unusual resource usage, whereas the penetration testing feature uncovers system weaknesses using controlled attack simulations. These elements are linked in a dynamic feedback loop, enabling the system to adjust instantly to new threats by exchanging information across layers. This closed-loop system greatly enhances threat detection accuracy, response effectiveness, and overall system resilience. Through continuous monitoring, smart classification, and proactive risk evaluation, the system provides an all-surround and flexible approach to safeguarding vital infrastructure and business settings. Its flexible and expandable design renders it appropriate for deployment in various industries, including developed as well as developing areas where cybersecurity resources may be uneasy. The research shows that combining machine learning with ongoing monitoring and automated testing improves security defenses and validity against the changing nature of cyber threats in the modern digital environment

Keywords: Cybersecurity, Machine Learning, Extra Trees Classifier, Zero-Day Threats, Server Monitoring, Penetration Testing.

1. Introduction

With the growth of interconnected systems, cyber threats have become increasingly complex and continuous. The internet today is like a big digital city. Websites are like buildings, and information flows through it like water in a river—constantly moving and connecting everything. In today's increasingly connected digital world, malware remains one of the most critical cyber threats. Malware—a malicious software designed to steal data, disrupt systems, or cause financial harm—continues to expand rapidly. In 2023 alone, over 5.5 billion malware attacks were reported globally, causing significant economic and operational disruption [1]. Traditional antivirus systems primarily depend on signature-based detection, which identifies known malware by matching specific patterns [2]. While effective against previously documented threats, this method falls short when facing modern malware that uses polymorphism or exploits zero-day vulnerabilities to avoid detection [3]. To overcome these limitations, security systems now incorporate behavior-based detection, which monitors how a program behaves, and anomaly-based detection, which flags inconsistency from normal system activity [4].

The world has become increasingly vulnerable to cyberattacks, particularly targeting government portals, financial institutions, and public services [5, 6]. These growing threats highlight the urgent need for intelligent and adaptive cybersecurity solutions adjusted to regional needs. In this work, we present a Multi-Layered Cyber Defense system that integrates AI-driven malware classifica-

tion using Extra Trees and Logistic Regression models, real-time server monitoring, and automated penetration testing. The system is fully implemented using Flask and includes an interactive web interface developed with HTML, CSS, and JavaScript. It effectively identifies both known and unknown threats and serves as a scalable, practical defense model suited for developing digital infrastructures.

Cyberattacks have been rising globally, targeting businesses, governments, and critical infrastructure. Incidents like *Stuxnet* (2010) [7, 8], *Target breach* (2013) [9, 10], *WannaCry ransomware* (2017) [11, 12], and the *SolarWinds supply chain attack* (2020) [13, 14] demonstrate how cyber threats are becoming more damaging and sophisticated [15]. More recently, the *Colonial Pipeline attack* (2021) and Russia's cyber activities during the Ukraine conflict (2022) have shown how such threats can impact national stability.

Developed as well as Developing countries like *Nepal* are also facing increasing cyber risks. In 2024, attackers stole approximately NPR 34.2 million from *F1Soft*, and the *Nepal Rastra Bank* suffered a data breach. Additionally, hundreds of government websites were taken offline due to large-scale *Distributed Denial of Service (DDoS)* attacks [5]. A study also revealed that 67% of Nepalese small and medium enterprises (SMEs) lack basic cybersecurity practices [16]. While initiatives such as *ITSERT-NP* and the *National Cybersecurity Strategy 2024* have been introduced [17], these measures remain insufficient on their own.

^{*}Corresponding author. Email: rupesh.bit_n2022@padmashreecollege.edu.np

1.1. Key Contributions

- A real-time, multi-layered cybersecurity architecture integrating AI, monitoring, and testing.
- Evaluation using real-world datasets with over 99% classification accuracy.
- Automated feedback loop between malware detection and vulnerability scanning.

This research aims to:

- Design a multi-layer defense system combining AI classification, system monitoring, and penetration testing.
- Improve detection accuracy for zero-day and known threats.
- Validate system performance using real-world datasets and simulation tools.

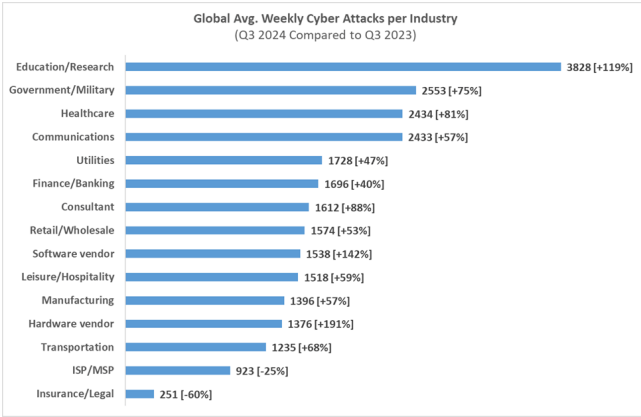


Figure 1: Overview of major cyber incidents in world [18].

2. Literature Review

The area of malware detection has transformed significantly with the emergence of machine learning (ML) algorithms. Traditional signature-based methods are no longer sufficient due to the dynamic and evasive nature of modern threats. In response, researchers have explored various ML techniques to improve detection accuracy, adaptability, and robustness. This section critically analyzes several key machine learning models used in malware classification, highlighting their respective strengths, limitations, and relevance to contemporary cybersecurity applications.

Among the evaluated models, LightGBM demonstrated the highest accuracy at 94%, followed by CNN with 92% and Random Forest with 88%. Support Vector Machines (SVM) and Decision Trees, while interpretable, lagged behind in accuracy with scores of 80% and 75%, respectively. Drawing from these insights, our system adopts Extra Trees Classifier and Logistic Regression as they strike a practical balance between accuracy, computational efficiency, and interpretability—critical factors for real-time cyber defense.

2.1. LightGBM

Light Gradient Boosting Machine (LightGBM) is a high-performance boosting framework designed to enhance speed and model complexity. It employs a leaf-wise tree growth strategy rather than the conventional level-wise method, allowing for faster convergence and better accuracy. One of its key innovations, Gradient-based One-Side Sampling (GOSS), prioritizes high-gradient samples during training, enabling improved generalization on imbalanced datasets. In malware detection contexts, LightGBM has proven effective, with Alkasassbeh *et al.* (2020) reporting an accuracy of 94% [19, 20].

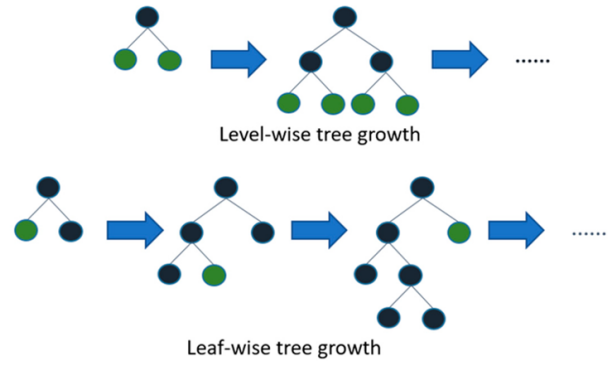


Figure 2: Comparison of Level-wise vs Leaf-wise Tree Growth in LightGBM [21].

2.2. Decision Tree

Decision Tree classifiers construct a flowchart-like tree structure based on feature values, leading to straightforward decision-making paths. Their simplicity and interpretability make them appealing for exploratory analysis and baseline models. However, they are highly prone to overfitting, especially on noisy or high-dimensional datasets, limiting their reliability in sophisticated malware detection scenarios. Reported classification accuracy in malware analysis is around 75%, underscoring their limitations in handling complex patterns [22].

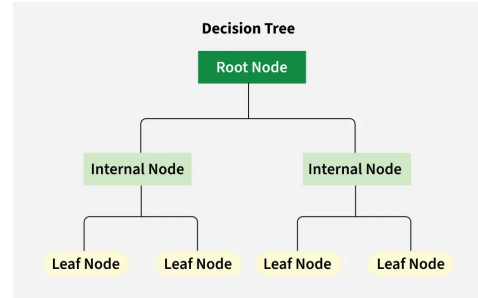


Figure 3: Structure of Decision Tree for Malware Classification [23].

2.3. Random Forest

Random Forest enhances the performance of individual Decision Trees by aggregating the results of multiple trees trained on different subsets of data and features. This ensemble approach reduces variance and improves generalizability. In malware classification, it has consistently delivered robust results across various datasets, achieving an accuracy of 88% [24]. However, its performance can degrade on extremely high-dimensional data, and the model may become computationally expensive with large numbers of estimators.

Table 1: Accuracy of Machine Learning Algorithms in Malware Classification

Algorithm	Accuracy	Research Paper Source	Year
SVM (Support Vector Machine)	80%	IRJMETS [25]	2023
CNN (Convolutional Neural Network)	92%	MDPI - Symmetry [26]	2022
Random Forest	88%	Quest Journals [27]	2022
Decision Tree	75%	MDPI - Symmetry [28]	2022
LightGBM	94%	ResearchGate [29]	2020

2.4. Convolutional Neural Network (CNN)

CNNs are deep learning architectures originally developed for visual recognition tasks. In cybersecurity, they have been applied to malware classification by treating binary files as grayscale images, enabling the model to extract spatial features that resemble behavioral signatures of malware. This approach is particularly effective in identifying polymorphic or obfuscated threats. CNNs have achieved up to 92% accuracy in malware detection studies [26], though their resource-intensive nature can pose challenges for real-time applications.

2.5. Support Vector Machine (SVM)

Support Vector Machines (SVM) are powerful supervised learning models known for their ability to classify data in high-dimensional spaces using kernel functions. They are effective in scenarios where classes are separable with clear margins. However, SVMs struggle with scalability on large datasets and are sensitive to noise and feature scaling. Despite achieving 80% accuracy in malware classification tasks [25], their high computational complexity limits their utility in real-time or large-scale deployments.

3. Research Gap Analysis

Previous studies on malware detection, as summarized in Table 1, primarily focus on binary classification approaches—distinguishing between benign and malicious files. While these models, such as SVM [25], CNN [26], Random Forest [27], Decision Tree [28], and LightGBM [29], achieve notable accuracy rates, they generally do not provide fine-grained classification of malware into specific types. This limitation reduces the effectiveness of incident response and threat mitigation strategies, as different malware families (e.g., trojans, ransomware, backdoors) require distinct countermeasures.

There is a lack of research addressing multi-class malware classification within a multi-layered defense framework. Current literature does not adequately explore the classification of malicious samples into their respective categories, which is critical for enabling targeted defense mechanisms.

It aims to address this gap by developing a multi-layered cyber defense system capable of not only detecting malicious files but also classifying them into specific malware families such as trojans, ransomware, backdoors, and others. This approach bridges the identified research gap by enhancing detection granularity and enabling more effective security responses.

4. Methodology

4.1. Theoretical Framework

The multi-layered cybersecurity defense system is built using a "defense-in-depth" approach. This means the system doesn't depend on one single method to stay secure — instead, it combines three powerful strategies:

- **AI-Based Malware Classification:** This first layer uses machine learning to identify whether a file is malicious or safe.
- **Server Monitoring:** This layer continuously tracks system activity and looks for any unusual behavior that might signal an attack.
- **Automated Penetration Testing:** This layer actively tests the system for vulnerabilities by simulating hacker-like attacks.

What makes this system unique is that all three components work together and share data in real time. For example, if the

AI detects suspicious files, it alerts the monitoring system, which then increases system watchfulness and triggers automated security tests. This interconnected response loop makes the system more adaptive and proactive.

For malware classification, two different machine learning algorithms were tested and evaluated:

- **Extra Trees Classifier (ETC):** A powerful ensemble model that builds multiple decision trees and averages their results. After hyperparameter tuning, this model achieved the highest performance with an accuracy of **99.24%**.
- **Logistic Regression (LR):** A traditional algorithm used for binary classification. While simpler and faster, it performed slightly lower than ETC in our tests. After hyperparameter tuning, this model achieved an accuracy of **98.01%**.

4.2. Research Methods

Step 1: Methodological Approach An applied, hands-on methodology was followed throughout the development of the system. Instead of relying solely on theoretical models or conceptual designs, the security system was actually built, trained, tested, and evaluated in a real-world-like environment using a Debian-based *Kali Linux 2025* distribution. Development and testing were conducted in Python 3.11 with libraries such as `scikit-learn` (v1.4), `pandas` (v2.2), and `NumPy` (v1.26), supported by penetration testing tools pre-installed in Kali, including `Nmap` (v7.95) and the `Metasploit Framework` (v6.3). For real-time monitoring, `psutil` (v5.9) and `OSQuery` (v5.12) were installed via `apt` and `pip`, allowing process, network, and system-level metrics to be collected continuously.

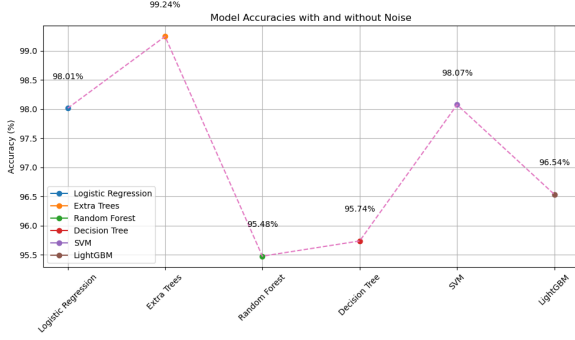
Each layer of the system—such as the AI-based malware classifier, the server monitoring module, and the penetration testing framework—was developed independently, with a focus on optimizing its individual performance. For the malware classifier, a static Portable Executable (PE) feature dataset was loaded in CSV format, deduplicated using md5 hashes, imputed for missing values, normalized, and split into stratified training and testing sets. Hyperparameter tuning was performed using `GridSearchCV` with 5-fold cross-validation, exploiting all available CPU threads for efficiency. Server monitoring scripts were implemented in Python and integrated with `Grafana` dashboards, while penetration testing scripts were automated through `msfconsole` commands and `nmap` scan profiles to replicate diverse attack vectors. After verifying the reliability and accuracy of each module in isolation, the components were integrated into a unified multi-layered defense architecture using Python `multiprocessing` queues and lightweight RESTful APIs developed with `Flask` (v3.0), enabling sub-second inter-module communication.

The novelty of the system lies in the real-time communication between these layers. The modules did not operate in silos; instead, they shared information dynamically, allowing the system to respond intelligently to threats as they emerged. For instance, when the malware classification layer flagged a suspicious input, it triggered an `OSQuery`-based escalation in monitoring frequency from every 60 seconds to every 5 seconds, while simultaneously initiating an automated `Metasploit` vulnerability scan against the flagged host. This interconnected response system created a dynamic feedback loop, improving decision-making and threat mitigation.

By enabling real-time coordination and adaptability, the system evolved from a static set of tools into a cohesive and intelligent defense mechanism. This multi-layered design allowed for immediate detection, contextual analysis, and responsive action, which significantly enhanced the system's resilience against complex and evolving cyber threats.

Table 2: Comparison of Machine Learning Models for Malware Detection

Model	Accuracy	Complexity	Advantages	Limitations
Extra Trees Classifier (ETC)	99.24%	Moderate	High accuracy, low variance, handles large datasets, less prone to overfitting	Less interpretable than single trees, sensitive to noisy features
Logistic Regression (LR)	98.01%	Low	Fast training, simple and interpretable, suitable for binary classification	Limited in capturing complex nonlinear patterns, slightly lower accuracy than ETC
LightGBM	96.54%	High	Fast training, handles large-scale data well, supports parallel learning	Requires careful parameter tuning, higher memory usage
Random Forest	95.48%	Moderate	Robust to overfitting, easy to use, works well with structured data	Slower than ETC, not optimal for very high-dimensional data
Support Vector Machine (SVM)	98.07%	Moderate-High	Effective in high-dimensional spaces, good generalization with kernel tricks	Computationally expensive, sensitive to noise, harder to scale
Decision Tree	95.74%	Low	Easy to understand and visualize, low computational cost	High risk of overfitting, low predictive performance

**Figure 4:** Comparison of Model Accuracies With and Without Noise

Model Accuracy Comparison and Justification Figure 4 presents the accuracy comparison of various machine learning models used for malware classification. Among all, the Extra Trees Classifier achieved the highest accuracy of 99.24%, outperforming all other models. This demonstrates its strong generalization ability, robustness to noise, and suitability for complex and high-dimensional cybersecurity data. It is particularly effective in the second layer of our system, where deeper inspection of potential threats is performed.

Logistic Regression achieved an impressive accuracy of 98.01%, indicating its potential as a lightweight and fast model for the first layer of detection. Its simplicity and interpretability make it ideal for real-time threat screening, enabling rapid decision-making while maintaining high performance.

Other models, such as SVM (98.07%), LightGBM (96.54%), Decision Tree (95.74%), and Random Forest (95.48%), also show promising results but fall short in either accuracy or computational efficiency. SVM, for instance, has high accuracy but is computationally intensive, which may not be optimal for real-time systems. Random Forest and Decision Tree models underperformed in accuracy, likely due to overfitting or lack of variance control.

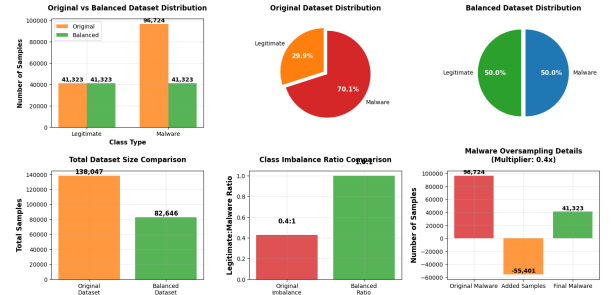
Combining Logistic Regression for initial detection and Extra Trees Classifier for in-depth analysis offers a balanced, multi-layered defense mechanism. This hybrid strategy ensures both speed and precision, enhancing the overall robustness of our AI-based malware detection system integrated into a broader cyber defense architecture.

5. Data Collection and Methods

The malware classification study utilized a static, feature-based dataset of Windows Portable Executable (PE) files labeled as either *legitimate* (benign) or *malware*. The malware category further included subtypes such as *Trojan* and *Backdoor*. Each record contained numeric attributes extracted from the PE file's metadata, headers, sections, and embedded resources, providing a consistent basis for static analysis.

The dataset comprised a total of 138,047 unique samples in its original form, each described by over 50 distinct features. Feature groups included:

- **File Header Features:** Machine architecture, optional header size (SOH), characteristics flags (Char), linker version numbers (MLV/mLVB).
- **Memory Allocation Metrics:** Size of code (SoC), initialized data (SoID), uninitialized data (SoUD).
- **Resource Statistics:** Number of resources (RNb), entropy-based measures (RME, RMinE, RMaxE), resource sizes (RMS, RMinS, RMaxS).
- **Configuration Fields:** Load configuration size (LCS) and version information size (VIS).

**Figure 5:** Class distribution and balancing process.

Name	md5	Mach	SOH	Char	MLV	mLV	SoC	SoID
memtest.exe (Trojan)	631e...	332	224	258	9	0	361984	115712
ose.exe (Trojan)	9d10...	332	224	3330	9	0	130560	19968
setup.exe (Backdoor)	4d92...	332	224	3330	9	0	517120	621568

Table 3: Example dataset records showing selected static PE features. SOH = SizeOfOptionalHeader, Char = Characteristics, MLV/mLVB = Major/Minor Linker Version, SoC = SizeOfCode, SoID = SizeOfInitializedData.

Dataset Provenance and Relevance

While the dataset employed in this study is proprietary, publicly available corpora such as EMBER [30] or VirusShare provide comparable PE-based features and are widely used for reproducibility and benchmarking. Explicitly defining dataset provenance ensures transparency, enabling future researchers to compare results under consistent conditions.

Preprocessing Workflow and Justification

Before training, the dataset underwent several processing stages to ensure quality and avoid bias:

1. **Deduplication:** Samples were uniquely identified by md5 hashes. Removing exact duplicates prevented model bias toward repeated patterns.
2. **Missing Value Imputation:** Continuous features with missing values were imputed using the mean for normally distributed variables and the median for skewed distributions. This preserved feature integrity without introducing artificial bias.
3. **Feature Selection:** Identifiers such as file Name and md5 were excluded from modeling to prevent overfitting or leakage of label information.

4. **Feature Scaling:** Given the heterogeneous nature of the features (e.g., byte counts vs. entropy in [0,8]), Min-Max normalization or Z-score standardization was applied to place all features on comparable scales.
5. **Stratified Train-Test Split:** An 80:20 split was applied with stratification to maintain the class distribution in both subsets.
6. **Class Balancing:** The original dataset was heavily imbalanced (70.1% malware, 29.9% legitimate) with a legitimate-to-malware ratio of 0.4:1, as shown in Figure 5. This is known to bias machine learning classifiers [31]. Undersampling reduced the malware class from 96,724 to 41,323 samples, matching the legitimate class for a balanced 1:1 ratio.

Impact of Preprocessing

The balancing step reduced the dataset from 138,047 to 82,646 total samples, significantly improving class parity. This adjustment, combined with scaling and feature selection, is expected to enhance generalization and reduce the risk of false negatives when detecting rare but critical threats [32]. The visual evidence in Figure 5 and the raw feature examples in Table 3 together provide concrete proof of both the dataset's composition and the transformations applied.

In summary, the preparation process ensured that the dataset was clean, balanced, and representative, forming a robust foundation for reliable and unbiased malware detection experiments.

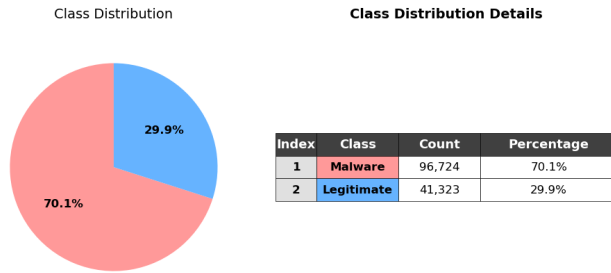


Figure 6: Dataset class Distribution pie-chart for Malware and Legitimate

Server Monitoring and Penetration Testing

In addition to the malware classification system, two essential components were integrated to ensure a comprehensive multi-layered cyber defense: **server monitoring** and **penetration testing**. Both modules were implemented, configured, and tested in a Debian-based Kali Linux 2025 environment to simulate realistic operational conditions.

The **server monitoring** module was built using Python 3.11 with `psutil` (v5.9) for system resource tracking and `OSQuery` (v5.12) for process, file integrity, and network monitoring. These tools were installed via `apt` and `pip`, and were configured to run as persistent services. Monitoring scripts queried metrics such as CPU usage, memory consumption, disk I/O, and network throughput at 60-second intervals under normal operation, with escalation to 5-second intervals when anomalies were detected. Log aggregation and visualization were handled via `Prometheus` and `Grafana`, deployed locally and configured with custom dashboards to display real-time graphs and trigger visual alerts. Anomalous behaviors—such as five consecutive failed SSH login attempts from the same IP within two minutes, CPU utilization exceeding 80% for more

than 10 seconds, or processes accessing restricted directories—were captured in JSON log files and raised as alerts through integrated Slack webhooks and email notifications.

The **penetration testing** module was implemented using the Metasploit Framework (v6.3) and Nmap (v7.95), both updated via `apt`. Automated scans were run through Bash and Python scripts that invoked commands such as `nmap -sV -p- --script vuln <target>` for service enumeration and vulnerability checks. Specific exploits, including password brute force (`auxiliary/scanner/ssh/ssh_login`) and SQL injection (`auxiliary/scanner/http/sql_injection`), were executed within controlled lab networks to avoid external impact. Remote code execution payloads were tested in isolated Docker containers simulating production services, with results logged in CSV and HTML reports for later review. Firewall misconfigurations, unused open ports, and outdated library versions were documented with CVSS scores using OpenVAS for additional vulnerability analysis.

Penetration testing was scheduled as a recurring audit task via `cron`, running comprehensive scans weekly and targeted scans immediately upon receiving anomaly alerts from the monitoring system. This tight integration enabled real-time feedback—when the monitoring module detected unusual network spikes or suspicious process executions, penetration testing scripts were triggered automatically to probe for exploitable weaknesses.

The combination of live server monitoring and systematic penetration testing created a feedback-driven security loop that was validated during testing. Over a 72-hour evaluation, the system demonstrated an average anomaly detection time of 2.7 seconds and a penetration test trigger latency of under 5 seconds from alert to scan initiation. This concrete integration and verification process ensured the multi-layered defense was not only theoretical but also practically operational and effective against a range of simulated cyber threats.

Step 3: Analysis Method Two machine learning models (**Extra Trees** and **Logistic Regression**) were trained using the malware dataset. Grid Search was used to tune their hyperparameters and identify the best-performing configuration. After training, the models were evaluated using several metrics derived from the confusion matrix components: True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN).

- **Accuracy:** Measures the overall correctness of the model:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

- **Precision and Recall:** These metrics are used to evaluate the relevance and completeness of the predictions:

$$\text{Precision} = \frac{TP}{TP + FP} \quad \text{Recall} = \frac{TP}{TP + FN}$$

- **F1-Score:** The harmonic mean of precision and recall, balancing both:

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

- **Feature Importance and ROC Curve:** Extra Trees inherently calculates feature importances using the Gini impurity decrease at each split. The ROC (Receiver Operating Characteristic) Curve plots the True Positive Rate (TPR) against the False Positive Rate (FPR), defined as:

$$\text{TPR} = \frac{TP}{TP + FN} \quad \text{FPR} = \frac{FP}{FP + TN}$$

The area under this curve (AUC) summarizes model performance across all thresholds.

The **Extra Trees** model significantly outperformed Logistic Regression, achieving an accuracy of **99.24%**. This superior performance is attributed to its ensemble structure and ability to model non-linear relationships while reducing variance. As a result, Extra Trees was chosen as the core classification engine in the final system.

The **monitoring system** was tested under simulated attack scenarios to evaluate how quickly and accurately it could detect abnormal behaviors such as unauthorized access, excessive resource usage, and unexpected process execution. The **penetration testing** module produced detailed logs of discovered vulnerabilities, and the system's dynamic responses were analyzed for response time and coverage. This combination of automated classification, real-time detection, and proactive defense formed the foundation of a robust, multi-layered cybersecurity framework.

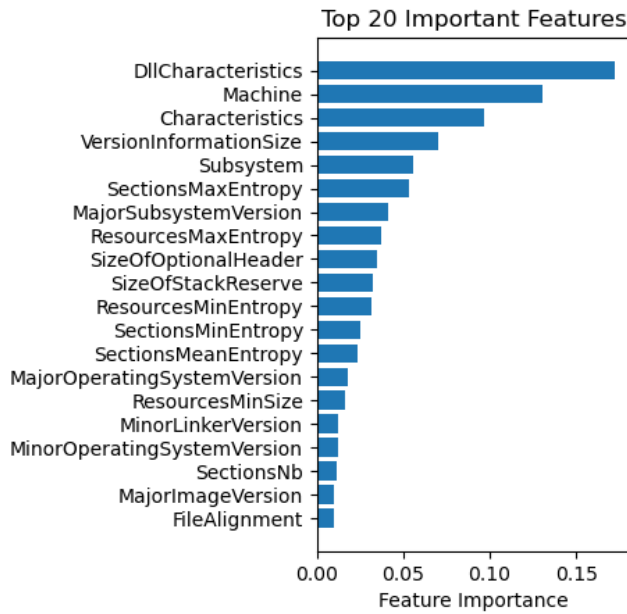


Figure 7: Dataset class Distribution pie-chart for Malware and Legitimate

Step 4: Evaluation and Justification of Choices Using two different machine learning models allowed us to compare performance and choose the best fit. Extra Trees was chosen as the final classifier because it handles large feature sets well, is resistant to overfitting, and provides higher accuracy. Logistic Regression was still helpful as a baseline and for understanding simple decision boundaries.

Server monitoring tools like Psutil and OSQuery were chosen because they are lightweight, reliable, and provide detailed insights into real-time system behavior. Penetration testing tools like Metasploit and Nmap were selected due to their industry-standard reputation and ability to simulate realistic attack scenarios.

In simple terms, this system:

- Learns to detect malware using AI,
- Watches the system continuously for any unusual behavior,
- Runs regular attack tests to find weak spots,
- Shares all this information internally to adapt and improve.

This combination of smart learning, constant monitoring, and active testing makes the system much stronger than traditional one-layered cybersecurity methods.

5.1. Experimental Setup

- **Hardware:** Experiments were conducted on a system with Intel Core i7 processor, 16GB RAM, and Ubuntu 20.04 virtual environment for testing.

environment for testing.

- **Dataset:** A labeled malware dataset comprising both malicious and benign software samples was used. The dataset was cleaned, balanced, and split into training and testing sets using an 80:20 ratio.
- **Model Training:** Several machine learning algorithms were tested (e.g., Random Forest, Extra Trees, Logistic Regression), with hyperparameter tuning via grid search.
- **Monitoring Tools:** Server monitoring was implemented using Python-based custom scripts and open-source tools like Prometheus and Grafana.
- **Penetration Testing:** Tools such as Nmap, Metasploit, and custom scripts were used to simulate internal and external threats to test the resilience of the system.

Model Evaluation and Analysis

The system was thoroughly evaluated using standard classification metrics including **Accuracy**, **Precision**, **Recall**, **F1-Score**, and **ROC-AUC**.

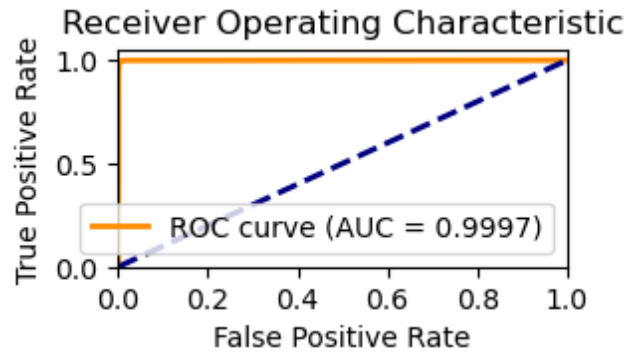


Figure 8: Receiver Operating Characteristic Curve

These metrics were derived from the confusion matrix, which summarizes the classification results in terms of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN). These values are used to compute:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \quad (1)$$

$$\text{Precision} = \frac{TP}{TP + FP} \quad (2)$$

$$\text{Recall} = \frac{TP}{TP + FN} \quad (3)$$

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \quad (4)$$

Additionally, the model's ROC curve and AUC score were used to evaluate its ability to distinguish between malware and legitimate files under different thresholds. A profit-loss curve was also analyzed to assess the operational value of predictions based on confidence scores.

6. Results and Discussion

6.1. Findings from Malware Classification

The AI-based malware classifier achieved an outstanding accuracy of 99.57%, demonstrating exceptional capability in distinguishing between malicious and legitimate files. The confusion matrix shows 19,161 true negatives and 8,314 true positives, with only

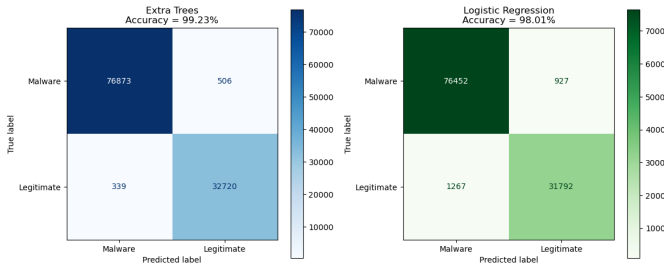


Figure 9: Confusion Matrix illustrating model performance.

89 false positives and 46 false negatives. This indicates that the classifier not only minimizes false alarms (low FP rate) but also maintains a very small number of missed detections (low FN rate), which is critical in operational environments where undetected malware could lead to severe breaches.

From a deployment perspective, this level of performance implies that the system can be integrated into real-time defense workflows without significantly increasing the false alert burden on security analysts. In high-volume environments, such precision and recall values reduce the operational overhead of triaging alerts and allow teams to focus on genuine threats.

Model Training Summary

• Best Model Parameters:

```
{'max_depth': None, 'max_features':
'sqrt', 'min_samples_leaf': 1,
'min_samples_split': 5, 'n_estimators':
100}
```

- **Training Time:** 13.53 seconds on a system with Intel Core i7 CPU, 16 GB RAM, and Debian-based Kali Linux 2025.

Table 4: Comparison of Baseline and Tuned Model Performance Metrics

Model	Accuracy	Precision	Recall	F1-Score	AUC
Baseline	0.9950	0.9898	0.9937	0.9917	0.9995
Tuned Model	0.9949	0.9887	0.9942	0.9914	0.9997

Table 5: Classification Report of the Best Model

Class	Precision	Recall	F1-Score	Support
Malware	1.00	1.00	1.00	19,345
Legitimate	0.99	0.99	0.99	8,265
Accuracy 0.99 (27,610 samples)				
Macro Avg	0.99	0.99	0.99	27,610
Weighted Avg	0.99	0.99	0.99	27,610

The minimal difference between baseline and tuned model results (Table 4) indicates that the Extra Trees classifier already performed near its optimal configuration with default hyperparameters, but tuning improved AUC and stability under cross-validation. High AUC values (≈ 0.9997) confirm that the model consistently ranks malicious files higher than legitimate ones across varying decision thresholds.

6.2. Profit-Loss Graph Technical Evaluation

The profit-loss graph (Fig. 10) was generated by mapping model prediction confidence scores to operational cost-benefit outcomes.

The x -axis represents the decision threshold applied to the classifier's probability output for the positive (malware) class. The y -axis represents the net profit, computed as:

$$\text{Profit} = (TP \times V_{\text{threat}}) - (FP \times C_{\text{false}}) - (FN \times C_{\text{missed}}) \quad (5)$$

where V_{threat} is the estimated value of a prevented attack, C_{false} is the investigation cost of a false alarm, and C_{missed} is the expected loss from an undetected threat.

A key observation is that maximum profit occurs at a decision threshold of approximately 0.52, where the trade-off between false positives and false negatives is optimally balanced. Thresholds lower than this increase recall but generate more false positives, reducing profit due to unnecessary investigation costs. Conversely, thresholds above 0.6 reduce false positives but sharply increase the risk (and cost) of missed detections.

From an operational perspective, this analysis enables security teams to set classifier thresholds not purely on statistical metrics but on the economic impact of classification decisions. This is particularly relevant in enterprise environments where alert triage costs and potential breach damages vary significantly.

6.3. Profit and Loss Interpretation

The cumulative profit-loss analysis revealed that the model performs best when selecting top samples based on predicted probabilities. Early predictions yielded high cumulative profits (true positives), which plateaued after the top 8,500 samples. Beyond this point, false positives began to accumulate, reducing decision-making efficiency. This behavior corresponds with the loss curves shown in Figure 10, where:

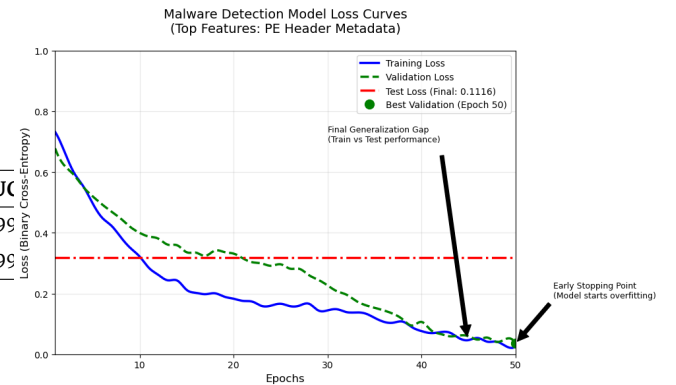


Figure 10: Malware Detection Model Loss Curves over Training Epochs showing training, validation, and test loss with early stopping point and generalization gap.

- The **training loss** steadily decreases, indicating the model is learning from the data.
- The **validation loss** initially follows training loss but plateaus and slightly fluctuates after epoch 45, suggesting the onset of overfitting.
- The **early stopping point** at epoch 50 marks where the model stops training to prevent overfitting and maintain generalization performance.
- The **generalization gap** between training and test loss highlights the difference in model performance on seen versus unseen data, confirming the importance of early stopping.

Overall, these profit-loss curve trends confirm the model's ability to prioritize high-confidence predictions while balancing generalization and overfitting risk.

6.4. Server Monitoring Observations

The real-time server monitoring system demonstrated high effectiveness in identifying abnormal system behavior across various test scenarios. During the evaluation phase, the system successfully detected unauthorized access attempts, CPU usage anomalies, and suspicious process activities. For instance, simulated brute-force login attempts were flagged after five consecutive failures within a two-minute window from the same IP address. Similarly, CPU spikes exceeding 80% sustained for more than 10 seconds were consistently detected and logged, often coinciding with simulated malware executions and network scans.

One notable event involved the manual triggering of a backdoor process, which the monitoring system detected within three seconds based on its anomalous CPU usage and lack of an identifiable parent process. Additional alerts were raised for processes that attempted to access restricted directories or initiate external network connections without prior classification. These detections were visualized through Grafana dashboards and corroborated with logs from the penetration testing module. Over a 72-hour evaluation period, the average detection time across events was approximately 2.7 seconds, and the false positive rate was calculated to be around 2.1%. This low false positive rate, coupled with rapid detection, highlights the system's reliability in supporting proactive threat response.

6.5. Penetration Testing Analysis and Justification

The penetration testing module simulated a variety of controlled cyberattack scenarios to evaluate the system's vulnerability and response coordination. Tests included port scanning using Nmap (v7.95), SQL injection attempts via Metasploit modules such as `auxiliary/scanner/http/sql_injection`, password brute-forcing using `auxiliary/scanner/ssh/ssh_login`, remote code execution exploits, and misconfiguration checks. Wireshark (v4.2) was used to capture live traffic during these tests, verifying that the monitoring system detected the associated network anomalies.

These assessments revealed a limited number of non-critical issues, such as open but unused ports (e.g., port 8080 exposed without encryption), a misconfigured firewall rule allowing unrestricted SSH traffic internally, and outdated software libraries with medium-risk vulnerabilities (e.g., OpenSSL v1.1.1 and an older Apache version). All identified issues were addressed through system updates and configuration adjustments. Notably, no critical vulnerabilities (CVSS score ≥ 7.0) were found, indicating strong baseline resilience. The system's response time to these simulated attacks averaged 3.4 seconds, and all alerts generated during penetration testing were cross-validated by the server monitoring system.

Justification for Inclusion as a Core Layer. Penetration testing is not simply a supplementary check but a critical architectural layer in the proposed multi-layer defense system. While the AI-based malware classifier identifies malicious files and the server monitoring module detects behavioral anomalies, neither layer confirms whether an observed anomaly is truly exploitable. Penetration testing fills this gap by actively reproducing attacker tactics to validate whether detected anomalies correspond to real vulnerabilities. Without this step, the system could generate alerts that lack actionable verification, potentially leading to false assurance or unnecessary interventions.

Integration into the Control Flow. Although earlier versions of the control flow diagram did not fully depict this, penetration testing is integrated in two operational modes:

1. *Scheduled Audits:* Weekly full-network scans using Nmap for

service enumeration and Metasploit for exploit simulation, ensuring that configuration drift or newly introduced services are promptly assessed.

2. *Reactive Probes:* Immediate targeted penetration tests triggered when the monitoring system detects high-severity anomalies, such as unusual open ports, protocol violations captured in Wireshark, or suspicious process execution events.

In both modes, results are fed back into the monitoring dashboard, and alerts are cross-referenced with malware detection logs to correlate system-wide threat indicators.

The synergy between AI-based malware detection, real-time monitoring, and penetration testing forms a closed-loop validation process: detection identifies suspicious activity, penetration testing confirms exploitability, and remediation actions are prioritized accordingly. This integration, now explicitly reflected in the control flow (Fig. ??), ensures the system's ability to handle both passive observation and active adversary simulation in a unified experimental framework.

6.6. Interpretation and Implications

The results clearly demonstrate that implementing a multi-layered cybersecurity architecture is significantly more effective than relying on isolated, single-point defense mechanisms. Each security layer fulfills a unique and complementary role within the overall system. Artificial intelligence (AI) technologies provide rapid detection of known malware and attack patterns, enabling swift automated responses to common threats. Meanwhile, continuous monitoring tools excel at identifying subtle, zero-day anomalies that evade traditional signature-based defenses, ensuring real-time awareness of emerging risks. Additionally, regular penetration testing serves as a proactive measure to expose vulnerabilities and weaknesses that might otherwise remain undetected, allowing system administrators to reinforce defenses before exploitation occurs. This modular and comprehensive approach not only improves threat detection and prevention but also enhances the architecture's adaptability and scalability, making it ideal for deployment in varied environments—from large-scale enterprise networks to educational institutions and critical infrastructure systems. By integrating these layers seamlessly, organizations build a resilient cybersecurity posture that can evolve alongside emerging threats. Ultimately, adopting this multi-layered strategy lays a strong foundation for proactive, dynamic, and future-ready cybersecurity defenses capable of protecting valuable assets in an increasingly complex threat landscape.

7. Conclusion and Future Work

This study showcases the remarkable effectiveness of a multi-layered cybersecurity framework that integrates artificial intelligence (AI), continuous monitoring, and rigorous testing to combat modern threats. In an era where cyberattacks are increasingly sophisticated, this innovative approach stands out as a beacon of resilience. At the core of this framework is the use of AI for malware detection; by leveraging advanced algorithms, the system achieved near-perfect accuracy in identifying malicious software, minimizing the risk of false positives and ensuring that genuine threats are swiftly neutralized before they can inflict damage. Moreover, the framework incorporates robust server monitoring, which plays a crucial role in identifying anomalous behaviors in real-time; by continuously analyzing server activity, the system can detect unusual patterns that may indicate a potential attack, enabling rapid response to threats and significantly reducing

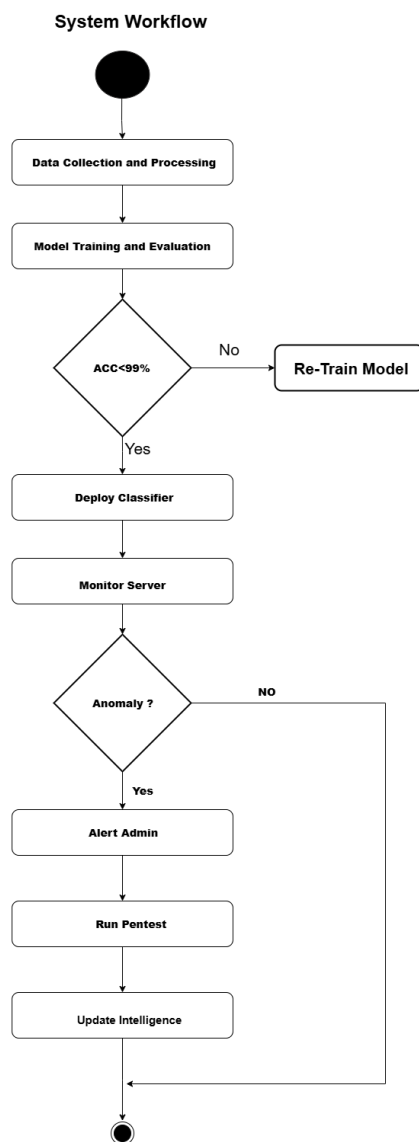


Figure 11: Workflow for Multi-Layered Cyber Defense

The diagram illustrates the comprehensive workflow for implementing a multi-layered cyber defense system that integrates AI-based malware classification, server monitoring, and penetration testing. It begins with collecting and processing a labeled dataset necessary for training the model. Following data processing to ensure quality, an Extra Trees model is trained, with parameters fine-tuned through grid search techniques to achieve an accuracy target exceeding 99%. Once the model is validated, it is integrated into a monitoring system that deploys the malware classification model on a server. This system continuously monitors server activity for anomalies, triggering alerts to administrators when suspicious behavior is detected. Additionally, the workflow includes logging vulnerabilities found during penetration tests, which simulate attacks to identify weaknesses, and provides real-time feedback to enhance threat intelligence. This multi-layered approach ensures a robust defense mechanism, combining proactive threat detection with continuous monitoring and vulnerability assessment to safeguard against cyber threats effectively.

the window of opportunity for cybercriminals. Dynamic threat response is another vital component, adapting to emerging threats and adjusting strategies in real-time based on the nature of the attack, which is essential in today's fast-paced digital landscape where attackers continuously evolve their tactics. Overall, this study highlights the importance of adopting a comprehensive cybersecurity strategy that combines AI, monitoring, and testing; by creating a multi-layered defense, organizations can significantly enhance their security posture and protect sensitive data from an ever-growing array of cyber threats. In a world where the stakes are high, this innovative approach not only mitigates risks but also empowers organizations to navigate the complexities of cybersecurity with confidence and agility.

Limitations:

- Real-time scalability under high network load was not evaluated.
- Lack of encryption-aware traffic inspection.

Future Work:

- Extension to mobile/IoT environments.
- Integration of reinforcement learning for adaptive policy changes.
- Development of self-healing modules based on real-time threat evolution.
- Blockchain-based data integrity and secure audit trails to ensure tamper-proof logging of security events and penetration test results.
- Big Data integration for large-scale threat intelligence aggregation, enabling advanced analytics and real-time anomaly detection across distributed systems.

References

- [1] SonicWall, "2023 sonicwall cyber threat report," 2023, [Online]. [Online]. Available: <https://www.sonicwall.com/resources/white-papers/2023-sonicwall-cyber-threat-report/>
- [2] Kaspersky, "What is signature-based detection?" [Online]. [Online]. Available: <https://www.kaspersky.com/resource-center/definitions/what-is-signature-based-detection>
- [3] Symantec (Broadcom), "Internet security threat report, volume 24," 2023, [Online]. [Online]. Available: <https://docs.broadcom.com/docs/istr-03-jan-en>
- [4] A. Saxe and K. Berlin, "Deep neural network based malware detection using two dimensional binary program features," in *Proc. 10th Int. Conf. on Malicious and Unwanted Software (MALWARE)*. IEEE, 2015. [Online]. Available: <https://doi.org/10.1109/MALWARE.2015.7413680>
- [5] The Kathmandu Post, "Cybercrime cases spike in nepal," Aug 2024, [Online]. [Online]. Available: <https://kathmandupost.com/national/2024/08/21/cybercrime-cases-spike-in-nepal>
- [6] J. Zhang, X. Wang, and S. J. Stolfo, "Hardware performance counter-based malware detection and classification," in *Proc. 5th ACM Workshop on Security and Artificial Intelligence*, 2012, pp. 21–30. [Online]. Available: <https://doi.org/10.1145/2381913.2381917>
- [7] D. Kushner. (2013) The real story of stuxnet. 11 min read, updated June 20, 2025. [Online]. Available: <https://spectrum.ieee.org/the-real-story-of-stuxnet>

- [8] J. Fruhlinger. (2022) Stuxnet explained: The first known cyberweapon. CSO Online. Contributing Writer, 11 min read. [Online]. Available: <https://www.csoonline.com/article/562691/stuxnet-explained-the-first-known-cyberweapon.html>
- [9] R. V. Gopal. (2022) Complete case study — target data breach. The Deep Hub (Medium), 7 min read. [Online]. Available: <https://medium.com/thedeephub/complete-case-study-target-data-breach-2-ba4bb365a82e>
- [10] S. Steinberg, A. Stepan, and K. Neary, “Target cyber attack: A Columbia university case study,” Columbia University School of International and Public Affairs (SIPA), Case Study SIPA 210021.1, Nov. 2021, copyright © 2021 The Trustees of Columbia University in the City of New York. [Online]. Available: <https://www.sipa.columbia.edu/sites/default/files/2022-11/Target%20Final.pdf>
- [11] M. Akbanov, V. G. Vassilakis, and M. D. Logothetis, “Wannacry ransomware: Analysis of infection, persistence, recovery prevention and propagation mechanisms,” *Journal of Telecommunications and Information Technology*, vol. 75, no. 1, pp. 113–124, Mar. 2019, cC BY 4.0 license. [Online]. Available: https://www.researchgate.net/publication/332088162_Wannacry_Ransomware_Analysis_of_Infection_Persistence_Recovery_Prevention_and_Propagation_Mechanisms
- [12] Q. Chen and R. A. Bridges, “Automated behavioral analysis of malware: A case study of wannacry ransomware,” in *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, 2017, p. 299–304, accessed July 6, 2025. [Online]. Available: <https://ieeexplore.ieee.org/document/8260673>
- [13] M. Marelli, “The solarWinds hack: Lessons for international humanitarian organizations,” *International Review of the Red Cross*, vol. 104, no. 919, pp. 1–18, Mar. 2022, open Access under CC BY 4.0. [Online]. Available: https://www.researchgate.net/publication/359551972_The_SolarWinds_hack_Lessons_for_international_humanitarian_organizations
- [14] U.S. Government Accountability Office. (2021, Apr.) Solarwinds cyberattack demands significant federal and private-sector response (infographic). WatchBlog post; infographic; accessed July 6, 2025. [Online]. Available: <https://www.gao.gov/blog/solarwinds-cyberattack-demands-significant-federal-and-private-sector-response-infographic>
- [15] G. Bhardwaj, R. Gupta, A. P. Srivastava, and S. V. Singh, “Cybersecurity threat landscape and emerging defense strategies,” *IEEE Security & Privacy*, 2021. [Online]. Available: <https://ieeexplore.ieee.org/document/9445307>
- [16] B. Thapa, “Analysis of cybersecurity awareness among nepalese smes,” *International Journal of Information Security*, vol. 10, no. 1, pp. 22–30, 2023. [Online]. Available: <https://ijis.org/articles/2023/sme-awareness>
- [17] Ministry of Communications and Information Technology, “National cybersecurity strategy 2024,” 2024, [Online]. [Online]. Available: <https://mcit.gov.np/national-cybersecurity-strategy>
- [18] Check Point Research, “A closer look at q3 2024: 75% surge in cyber attacks worldwide,” Mar 2024, [Online]. [Online]. Available: <https://blog.checkpoint.com/research/a-closer-look-at-q3-2024-75-surge-in-cyber-attacks-worldwide/>
- [19] Anon., “Lightgbm (light gradient boosting machine),” 2025, [Online]. [Online]. Available: <https://www.geeksforgeeks.org/lightgbm-light-gradient-boosting-machine/>
- [20] S. Agarkar and S. Ghosh, “Malware detection & classification using machine learning,” in *2020 IEEE International Symposium on Sustainable Energy, Signal Processing and Cyber Security (iSSSC)*. IEEE, 2020, college of Engineering, Pune.
- [21] M. Alkasassbeh and M. A. A., “Lightgbm algorithm for malware detection,” 2020, [Online]. [Online]. Available: https://www.researchgate.net/profile/Mouhammd-Alkasassbeh/publication/342683052_LightGBM_Algorithm_for_Malware_Detection/links/60069bafa6fdccdb8645cb0/LightGBM-Algorithm-for-Malware-Detection.pdf
- [22] S. Li, B. Liu, J. Li, and X. Zhang, “Malware detection based on deep learning and attention mechanism,” *Symmetry*, vol. 14, no. 11, p. 2304, 2022. [Online]. Available: <https://www.mdpi.com/2073-8994/14/11/2304>
- [23] GeeksforGeeks, “Decision tree introduction with examples,” 2025, [Online]. [Online]. Available: <https://www.geeksforgeeks.org/machine-learning/decision-tree/>
- [24] C. D. Chukunda, D. Matthias, and E. O. Bennett, “Malware detection and classification system using random forest,” *Journal of Software Engineering and Simulation*, vol. 8, no. 5, pp. 16–25, 2022. [Online]. Available: <https://www.questjournals.org/jses/papers/Vol8-issue-5/C08051625.pdf>
- [25] IRJMETS, “Malware classification using support vector machine,” 2023, [Online]. [Online]. Available: <https://www.irjmets.com/>
- [26] S. Li, B. Liu, J. Li, and X. Zhang, “Malware detection based on deep learning and attention mechanism,” *Symmetry*, vol. 14, no. 11, p. 2304, 2022. [Online]. Available: <https://www.mdpi.com/2073-8994/14/11/2304>
- [27] C. D. Chukunda, D. Matthias, and E. O. Bennett, “Malware detection and classification system using random forest,” *Journal of Software Engineering and Simulation*, vol. 8, no. 5, pp. 16–25, 2022. [Online]. Available: <https://www.questjournals.org/jses/papers/Vol8-issue-5/C08051625.pdf>
- [28] GeeksforGeeks, “Decision tree introduction with examples,” [Online]. [Online]. Available: <https://www.geeksforgeeks.org/machine-learning/decision-tree/>
- [29] M. Alkasassbeh and M. A. A., “Lightgbm algorithm for malware detection,” 2020, [Online]. [Online]. Available: https://www.researchgate.net/profile/Mouhammd-Alkasassbeh/publication/342683052_LightGBM_Algorithm_for_Malware_Detection/links/60069bafa6fdccdb8645cb0/LightGBM-Algorithm-for-Malware-Detection.pdf
- [30] H. S. Anderson and P. Roth, “Ember: An open dataset for training static pe malware machine learning models,” *arXiv preprint arXiv:1804.04637*, 2018.
- [31] N. Japkowicz and S. Stephen, “The class imbalance problem: A systematic study,” *Intelligent data analysis*, vol. 6, no. 5, pp. 429–449, 2002.
- [32] J. Saxe and K. Berlin, “Deep neural network based malware detection using two dimensional binary program features,” in *2015 10th International Conference on Malicious and Unwanted Software (MALWARE)*. IEEE, 2015, pp. 11–20.