Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Malware Detection, Classification and Mitigation Using Machine Learning Algorithms: A Comparative Study on PE Header, Temporal and API Call Datasets with a Risk Based Quarantine Policy

Nnaemeka Ugwumba

nnaemeka_ugwumba@uniport.edu.ng

University of Port Harcourt https://orcid.org/0009-0000-2493-9846

Clement Barisua Ndeekor

clement.ndeekor@uniport.edu.ng

University of Port Harcourt, Port Harcourt, Nigeria

Ubuara Precious Oghenero

University of Port Harcourt, Port Harcourt, Nigeria

Research Article

Keywords: malware detection, malware classification, malware mitigation, machine learning, Windows PE files, static analysis, Random Forest, XGBoost, LightGBM, logistic regression, concept drift, API call sequences, malware family classification, quarantine policy, false positive rate, cybersecurity, Nigeria

Posted Date: November 24th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11816869/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: The authors declare no competing interests.

Malware Detection, Classification and Mitigation Using Machine Learning Algorithms: A Comparative Study on PE Header, Temporal and API Call Datasets with a Risk Based Quarantine Policy

Clement Barisua Ndeekor
clement.ndeekor@uniport.edu.ng
Department of Computer Science, Faculty of Computing
University of Port Harcourt, Port Harcourt, Nigeria

Nnaemeka Kingsley Ugwumba
nnaemeka_ugwumba@uniport.edu.ng
Department of Computer Science, Faculty of Computing
University of Port Harcourt, Port Harcourt, Nigeria

Ubuara Precious Oghenero
Department of Computer Science, Faculty of Computing
University of Port Harcourt, Port Harcourt, Nigeria

Abstract

Malware detectors built with machine learning often report accuracy above 99 percent, yet such figures rarely say how the detector will perform on next year's malware or what should happen to a flagged file. This paper presents CyberGuard, a web based malware detection, classification and mitigation system developed at the University of Port Harcourt, Nigeria, which parses Windows Portable Executable (PE) files statically without executing them, converts them to 2,381 EMBER features, scores them with a machine learning model, maps malicious files to a category, recommends a response and can lock suspicious files in a read only quarantine vault. We then evaluate the methods the system depends on with three public datasets. On 5,207 files of the ClaMP PE header dataset, ten algorithms were compared with five fold cross validation: Random Forest, LightGBM and XGBoost reached about 99.2 percent accuracy with a ROC AUC of 0.999 and false positive rates near 1.1 percent, while logistic regression, the algorithm in the deployed engine, reached 96.0 percent with 4.3 percent false positives. A single feature audit showed that the header checksum alone separates the classes with a ROC AUC of 0.81, a shortcut that reflects how benign files were collected. On 43,641 files of the Brazilian malware dataset collected from 2013 to 2020, a random split suggested that LightGBM detects 97.8 percent of malware at a 1 percent false positive rate, but a model frozen after 2013 detected only 76.3 percent on average from 2015 to 2020, and logistic regression fell to 59.0 percent by 2020; yearly retraining raised the averages to 86.1 and 78.8 percent. On the Mal API 2019 dataset, after removing 643 duplicate and 48 conflicting API call sequences, eight algorithms assigned malware to eight families with at most 65.5 percent accuracy (LightGBM, macro F1 0.667). We propose a three band mitigation policy that quarantines files above a threshold set at a 1 percent benign false positive rate, sends uncertain files to an analyst and allows the rest, together with a family based response playbook aligned with NIST guidance and MITRE ATT&CK, and we give recommendations for replacing the deployed engine, reporting accuracy on time ordered data and retraining regularly.

Keywords: malware detection, malware classification, malware mitigation, machine learning, Windows PE files, static analysis, Random Forest, XGBoost, LightGBM, logistic regression, concept drift, API call sequences, malware family classification, quarantine policy, false positive rate, cybersecurity, Nigeria

1 Introduction

Malware is software written to damage a computer, steal its data or take control of it. Viruses attach to legitimate programs, worms spread across networks on their own, Trojans hide inside programs that look useful, spyware collects information without consent, and ransomware encrypts files and demands payment. Banks, hospitals, universities and government agencies in Nigeria now run on networked computers, and each of these kinds of malware threatens the confidentiality, integrity or availability of the data they hold.

For decades the main defence has been signature matching: an antivirus product keeps a database of byte patterns or hashes taken from known malware and flags any file that matches. Signatures are fast and precise for threats that have already been seen, but a new sample, or an old one that has been repacked, recompiled or encrypted, has no signature until an analyst writes one. Machine learning offers a different approach. A model trained on many labelled files learns which properties of a file separate malware from benign software, and it can score a file it has never seen [12, 15].

Published results for machine learning detectors are often very high, frequently above 99 percent accuracy. Arp et al. [2] and Pendlebury et al. [26] showed that many such results depend on how the data were split. When training and test files are drawn at random from the same period, the test set contains close relatives of the training files, and the score overstates how the detector will perform on the malware that arrives next month. A second gap lies between detection and response: most studies stop at a label, while an organisation needs to know what to do with a flagged file, how many files will be wrongly quarantined, and how many will need an analyst.

This paper reports the design of CyberGuard, a web application built at the University of Port Harcourt that

analyses Windows Portable Executable (PE) files statically, scores them with a machine learning model, assigns a category and recommends mitigation, and an evaluation of the methods it depends on, carried out on three public datasets. We ask four questions.

1. How should a safe, web based malware detection, classification and mitigation system be designed, and how was CyberGuard built?

2. How do ten machine learning algorithms compare for detecting malware from static PE header features, and which features drive their decisions?

3. How much does detection accuracy fall when a model trained on older malware meets newer malware, and how much does regular retraining recover?

4. How accurately can malware be assigned to families from its API call behaviour, and how should detection scores and family labels be turned into mitigation actions?

Our contributions are:

1. the design and implementation of a static, non executing malware analysis system with a FastAPI backend, an EMBER feature vectoriser, a scikit learn detection engine, a read only quarantine vault and a category based mitigation module;

2. a comparison of ten algorithms on the ClaMP PE header dataset with five fold cross validation, confidence intervals, false positive and false negative rates, timing, feature importance and a single feature audit that exposes dataset shortcuts;

3. a time ordered evaluation on 43,641 unique files from the Brazilian malware dataset, collected from 2013 to 2020, comparing a random split with a model frozen after its first year and with yearly retraining;

4. an eight family classification study on the Mal API 2019 dataset of API call sequences with eight algorithms, after removing duplicated and conflicting samples;

5. a three band mitigation policy (quarantine, analyst review, allow) with thresholds set from benign files, evaluated on held out data and over time, and a family based response playbook aligned with NIST guidance and MITRE ATT&CK.

Section 2 reviews related work. Section 3 describes the system. Section 4 sets out the experimental method. Section 5 presents the results, Section 6 discusses them, and Section 7 concludes.

2 Related Work

2.1 Analysis approaches and features

Static analysis inspects a file without running it: header fields, section names and sizes, byte entropy, imported libraries and functions, and printable strings. Dynamic analysis runs the file in a sandbox and records what it does, most often the sequence of Windows API calls. Damodaran et al. [12] compared static, dynamic and hybrid features with hidden Markov models and found that dynamic features generally gave the best detection, while static features were cheaper and safer to collect. Static features are what a web service that must never execute an uploaded file can use, and dynamic API traces are what is needed to tell families apart by behaviour.

Datasets shape the field. Kumar et al. [19] released ClaMP, a set of 5,184 PE files described by raw header values and derived features such as section entropy and packer type, and reported accuracy near 99 percent with Random Forest. Anderson and Roth [1] released EMBER, 1.1 million PE files described by a 2,381 element feature vector, together with a gradient boosted baseline; CyberGuard uses the EMBER vectoriser. Yang et al. [36] added BODMAS, with family labels and timestamps, to support temporal studies. Ceschin et al. [8] collected malware targeting Brazilian users over several years and showed that classifiers lose accuracy as the malware changes. Catak and Yazı [6] and Catak et al. [5] released Mal API 2019, API call sequences of 7,107 malware samples in eight families recorded in a Cuckoo sandbox. Ronen et al. [29] described the Microsoft Malware Classification Challenge, a nine family benchmark built from disassembly and byte content.

2.2 Malware categories and the CIA triad

The design of CyberGuard’s mitigation module rests on the confidentiality, integrity and availability (CIA) triad, the three properties that information security protects. Each category of malware attacks one or more of them, and the property under attack decides the first response. Table 1 summarises the categories considered in this study. Spyware and backdoors threaten confidentiality, because their purpose is to read data or give an outsider access; the response must therefore include changing credentials and revoking sessions, since the data may already have left the machine. Viruses, droppers and downloaders threaten integrity, because they change files or install other programs; the response must find and remove everything they wrote. Worms and ransomware threaten availability, because they spread across a network or lock files; the response must isolate the machine quickly and restore from backups. A Trojan can do any of these, depending on its payload.

Table 1. Malware categories, how they spread and the security property mainly at risk

Category	Behaviour	CIA impact
Virus	attaches to programs, runs when the host runs	integrity
Worm	spreads by itself through network services	availability, integrity
Trojan	hides inside a program that looks useful	all three
Backdoor	gives an outsider remote access	confidentiality, integrity
Spyware	records keystrokes, screens or files	confidentiality
Adware	shows unwanted advertising, changes browser	confidentiality
Downloader	fetches further malware from the internet	integrity
Dropper	carries and installs a hidden payload	integrity
Ransomware	encrypts files and demands payment	availability

2.3 Algorithms

Tree ensembles dominate studies on tabular PE features. Random Forest [3] averages many decision trees grown on bootstrap samples, while XGBoost [9] and LightGBM [18] fit trees in sequence to the remaining errors. Louk and Tama [20] compared tree based ensembles on PE malware datasets and found gradient boosting and Random Forest consistently at the top. Roy et al. [30] stacked several learners with feature engineering to detect obfuscated malware, and Dabas et al. [11] combined filter and wrapper feature selection before classification. Manzano et al. [21] and Hossain and Islam [16] reported that feature selection followed by ensemble learning gave the best results for network malware and botnet traffic. Deep networks have been trained on raw bytes [28], on byte histograms and string features [31], and on bytes rendered as images [22]; Deldar and Abadi [13] and Gopinath and Sethuraman [15] survey these methods. Brown et al. [4] showed that automated search can tune such networks. Vapnik's statistical learning theory [34] explains why models must generalise beyond their training sample, and the no free lunch theorem [35] why no algorithm is best for every dataset, which is the reason we compare many.

2.4 Evaluation pitfalls and drift

Arp et al. [2] listed ten pitfalls of machine learning in security, among them sampling bias, data snooping, spurious correlations and inappropriate baselines. Pendlebury et al. [26] showed that random splits of Android malware data inflated F1 scores and proposed that training files must precede test files in time. Jordaney et al. [17] proposed statistical tests to flag predictions on drifted samples, Chen et al. [10] studied continuous learning schedules for Android malware, and Ceschin et al. [7] modelled malware detection as an evolving data stream. Peppes et al. [27] generated synthetic zero day samples with generative networks to test classifiers, and Galli et al. [14] studied explanations for behavioural detectors. Nazir et al. [23] reviewed the same issues in IoT botnet detection.

2.5 From detection to mitigation

The NIST guide to malware incident prevention and handling [32] divides response into containment, eradication and recovery, and the current NIST incident response recommendations [24] place these inside the Cybersecurity Framework 2.0 functions. MITRE ATT&CK [33] catalogues adversary techniques and links each to mitigations. Few machine learning papers connect a detector's score to these actions or measure how many files each action would touch.

2.6 Research gap

Table 2 places this study among the closest work. Most studies report one dataset, one random split and one label. This paper evaluates the same family of algorithms on three datasets that test three different abilities (static detection, detection over time, and family classification), audits the data for duplicates and shortcuts, and turns scores into a measured mitigation policy inside a working system.

Table 2. Prior work closest to this study

Work	Data	What it provides	What it does not address
Kumar et al. [19]	ClaMP PE headers	integrated feature set, Random Forest near 99 percent	time split, shortcut audit, response
Anderson and Roth [1]	EMBER, 1.1 million PE files	2,381 feature vectoriser, boosted baseline	family classification, mitigation
Ceschin et al. [7, 8]	Brazilian malware, 2013 to 2020	evidence of drift, stream learning	comparison with deployed linear model, quarantine policy
Catak et al. [5, 6]	Mal API 2019	API sequence dataset, LSTM and CNN models	duplicate removal, classical baselines
Pendlebury et al. [26], Arp et al. [2]	Android and others	evaluation pitfalls, time aware splits	Windows PE, mitigation
Louk and Tama [20], Roy et al. [30]	PE datasets	tree ensembles, stacking	drift, operating thresholds

3 System Design and Implementation

3.1 Requirements and pipeline

CyberGuard was designed to accept a PE file (exe, dll, sys or bin) of up to 50 MB, decide whether it is malicious without ever executing it, assign a category to malicious files, recommend a response, optionally quarantine the uploaded copy, and keep a history of analyses. Figure 1 shows the pipeline proposed in the design stage: data collection, preprocessing, feature extraction, model training and comparison, detection, classification and mitigation recommendation. The functional requirements were file upload and validation, static parsing, feature extraction, detection, classification, mitigation guidance, quarantine, history and a model benchmark page. The main nonfunctional requirement was safety: an uploaded file must never run.

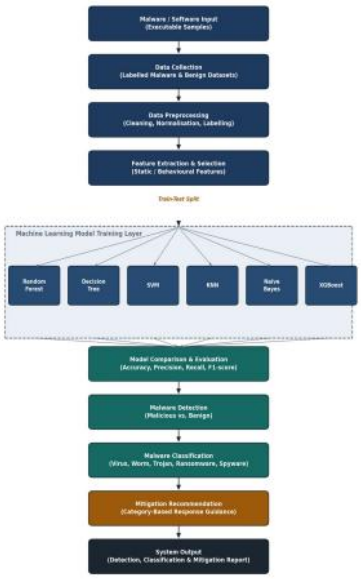

Figure 1. Proposed pipeline for malware detection, classification and mitigation.

Figure 1 also shows how the three tasks depend on each other. Classification runs only on files the detector has called malicious, and mitigation takes both the detection score and the family as inputs. An error at the detection stage therefore propagates: a missed file receives no mitigation, and a benign file wrongly flagged receives a response it does not need. This is why Section 5 measures detection errors at fixed false positive rates rather than only overall accuracy.

3.2 Architecture

Figure 2 shows the implemented architecture. The client is a Next.js and React application styled with Tailwind CSS and hosted on Vercel. It calls a versioned REST interface provided by a Python FastAPI service with Pydantic validation. Inside the service, an upload module checks file size, type and path, a static PE engine built on pefile and LIEF parses the headers, sections, imports and exports, and an EMBER vectoriser converts the parsed file into a 2,381 element numeric vector [1]. A scikit learn [25] logistic regression model scores the vector, a category mapping module assigns a category, and a mitigation module returns guidance. Records are stored in MongoDB through the asynchronous Motor driver. Files flagged as malicious can be moved to a quarantine vault where their permissions are set to read only for the owner (mode 0400), so they cannot be executed or modified.

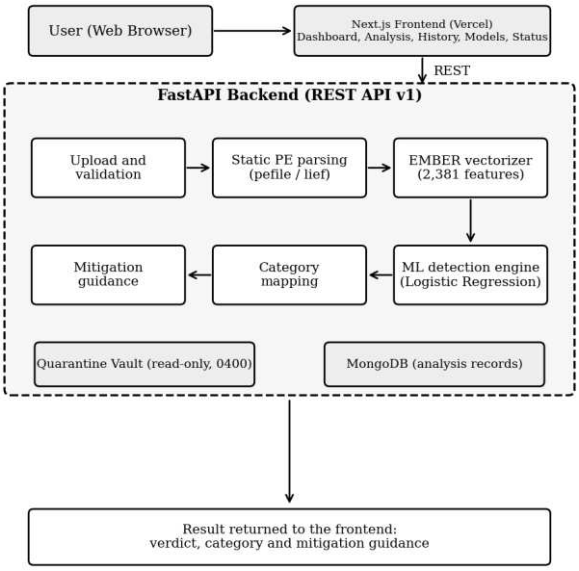

Figure 2. Architecture of CyberGuard: Next.js frontend, FastAPI backend with static PE parsing, EMBER vectoriser, detection engine, category mapping, mitigation guidance, quarantine vault and MongoDB.

The architecture in Figure 2 keeps all analysis on the server. The browser only uploads the file and displays the result, so the model and the quarantine vault are never exposed to the client. Static parsing is the central safety decision: because pefile and LIEF read the file as data, a malicious sample cannot run during analysis, and the server needs no sandbox. The cost of this choice is that CyberGuard sees only what the file declares about itself, not what it does, which limits family classification (Section 5.3).

3.3 Workflow

Figure 3 shows the workflow. The user opens the analysis page, selects a file and may tick the automatic quarantine option. The backend validates the file; an invalid file produces an error and ends the process. A valid file is parsed, vectorised and scored. A benign verdict is reported and saved. A malicious verdict is mapped to a category, the file is quarantined if the option was ticked, and the verdict, category and mitigation guidance are displayed and saved.

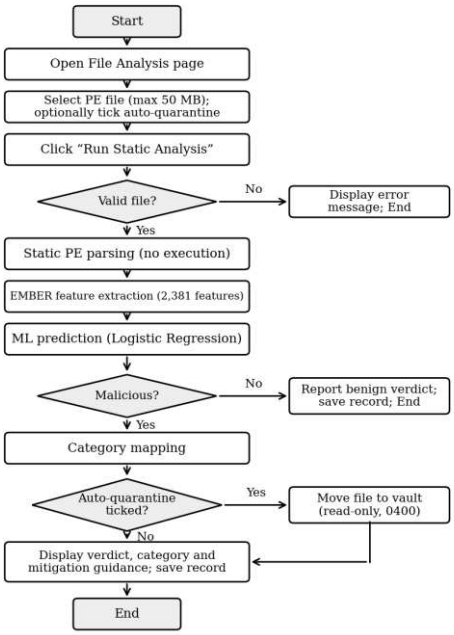

Figure 3. Workflow of CyberGuard from file selection to verdict, category, mitigation guidance and optional quarantine.

The flowchart in Figure 3 has two decision points that matter for mitigation. The first, *Malicious?*, is a threshold on the model's probability, which the deployed system sets at 0.5. The second, *Auto quarantine ticked?*, leaves the response to the user. Section 5.4 replaces both with a policy in which the threshold is set from a target false positive rate and a middle band of uncertain files is sent to an analyst instead of being quarantined or released.

3.4 User interface

Figure 4 shows the dashboard. Five tiles report total scans, malicious detections, benign binaries, quarantined files and the F1 score of the active model. Below them, a classification distribution panel and a risk severity breakdown (critical, high, medium, low) summarise past analyses, and a production model panel names the algorithm (logistic regression), its test accuracy, its test F1 score and the 2,381 feature dimensions. The counters read zero because the screenshot was taken on a fresh deployment. The panel reports a test accuracy of 100 percent and an F1 score of 1.0; the training data and split behind these figures were not recorded in the project files, and Section 5 shows why such a figure should not be read as the accuracy users will see on new malware. The classification panel is labelled heuristic, which means that categories in the deployed system are assigned by rules, not by a trained model; Section 5.3 tests whether a trained model can do this job.

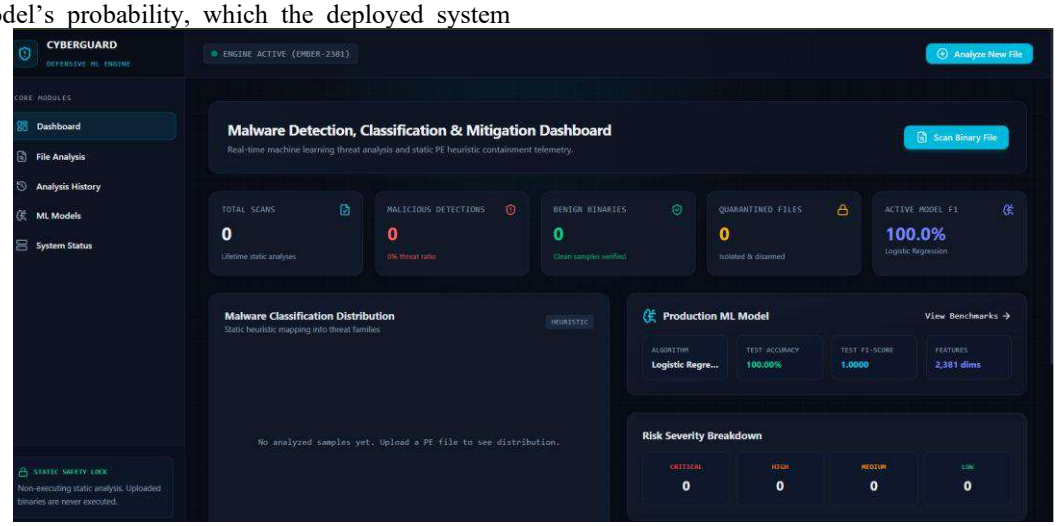

Figure 4. CyberGuard dashboard with scan counters, active model F1 score, classification distribution, production model panel and risk severity breakdown.

Figure 5 shows the Static Binary Threat Analyzer, where analysis starts. The page carries a safety banner stating that uploaded files are parsed structurally and never executed, a drag and drop area that accepts exe, dll, sys and bin files up to 50 MB, a checkbox to quarantine the sample automatically if it is detected as malicious, and the Run Static Analysis button.

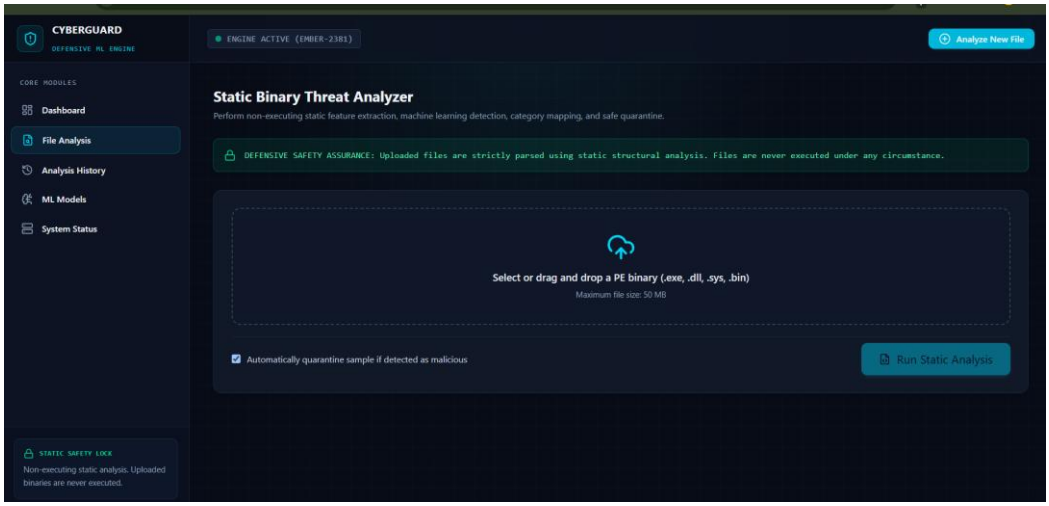

Figure 5. Static Binary Threat Analyzer: upload area, 50 MB limit, safety banner and automatic quarantine option.

Figure 6 shows the Analysis Investigation Log. Past analyses can be searched by file name or SHA256 hash and filtered by verdict, risk level and quarantine state, with paging. An auditable history matters for incident response, because it lets an analyst find every machine on which a file now known to be malicious was scanned earlier.

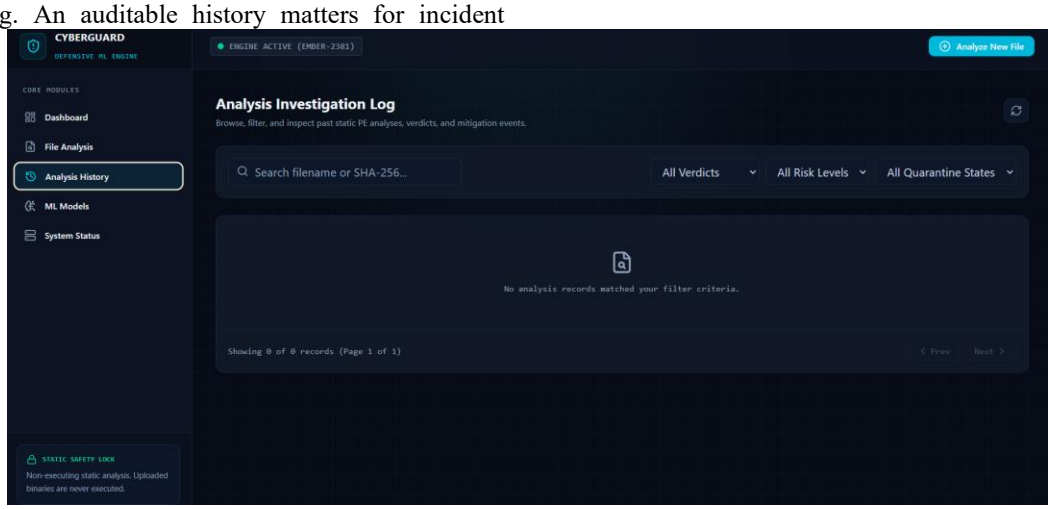

Figure 6. Analysis Investigation Log with search by file name or hash and filters for verdict, risk level and quarantine state.

Figure 7 shows the System Status page. It reports the state of four services (the FastAPI core, the machine learning engine with logistic regression loaded, the MongoDB store and the active quarantine vault) and lists the defensive principles of the design: no binary is run or spawned as a child process, upload and quarantine paths are checked against their parent directories to prevent path traversal, and quarantined files have their execution bits removed and are locked to read only. It also lists the technology stack.

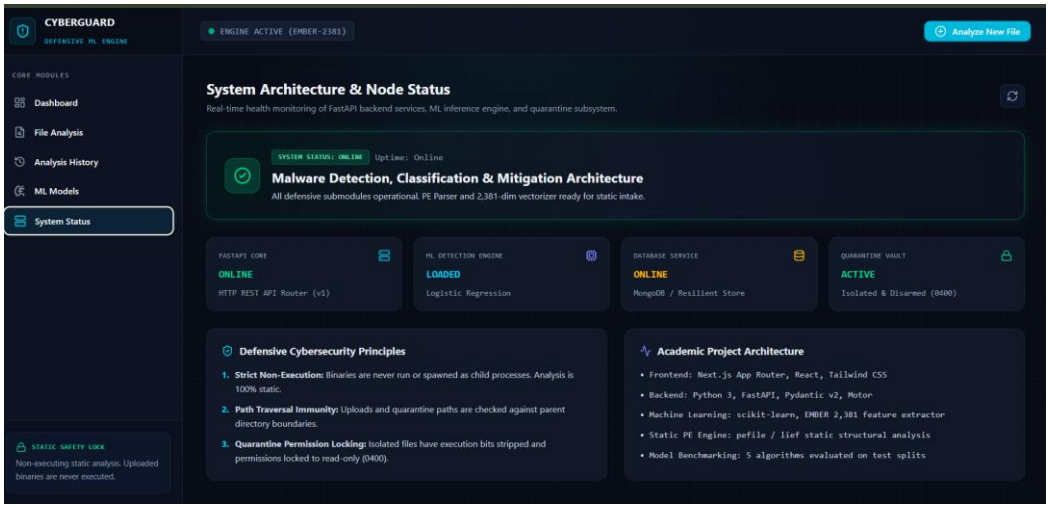

Figure 7. System Status page: service health, defensive principles (non execution, path traversal protection, quarantine permission locking) and technology stack.

Figure 8 shows the analyzer after a request failed because the backend could not be reached. The message *Failed to fetch* appears below the upload area and no verdict is produced. This is the correct behaviour (the system does not guess when it cannot analyse), but the message tells the user nothing about the cause. A clearer message, separating an unreachable server from an invalid or oversized file, is one of the improvements listed in Section 6.



Figure 8. Error state of the analyzer when the backend cannot be reached.

Table 3 summarises the modules, their implementation and the safeguards each provides.

Table 3. CyberGuard modules and safeguards

Module	Implementation	Safeguard or output
Upload	FastAPI, Pydantic	50 MB limit, type check, path traversal check
Static parser	pefile, LIEF	file read as data, never executed
Vectoriser	EMBER	2,381 element vector
Detection	scikit learn logistic regression	probability of malware
Category	rule based mapping	category label
Mitigation	guidance by category	advice shown with verdict
Quarantine	server vault	execution bits removed, mode 0400
History	MongoDB, Motor	searchable record by hash

4 Experimental Method

The deployed system records neither its training data nor its evaluation split, so its own figures cannot be checked. We therefore evaluated the methods it depends on with three public datasets, each chosen to answer one question (Table 4). All experiments were run in Python with scikit learn 1.9 [25], XGBoost 3.4 [9] and LightGBM 4.7 [18] on a two core machine with 7 GB of memory, with a fixed random seed of 42. The whole study runs from one script in about 40 minutes.

Table 4. Datasets used in the evaluation

Dataset	Content	Samples used	Question
ClaMP [19]	PE header and derived features	5,207 (2,719 malware)	static detection, ten algorithms
Brazilian [8]	PE headers, imported DLLs and functions, dated	43,641 (22,546 malware)	detection over time, 2013 to 2020
Mal API 2019 [6]	Windows API call sequences	6,361 in 8 families	family classification

4.1 Algorithms compared

Ten algorithms were used for detection in Study 1 and eight for family classification in Study 3. Together they span the main families of supervised learning, so that the

comparison shows which kind of decision rule suits malware data rather than which tuned model wins.

Logistic regression models the probability that a file is malware as p = 1/(1+e^{−(w^x+b)}), where x is the feature vector, w a weight per feature and b a bias, and fits w and b by minimising the cross entropy loss with an L2 penalty. Its decision boundary is a hyperplane, so it cannot represent interactions between features unless they are built into x. It is the algorithm in CyberGuard's deployed engine.

Gaussian naive Bayes applies Bayes' rule under the assumption that features are independent given the class and normally distributed within each class. It trains in a single pass and serves as a lower baseline.

k nearest neighbours labels a file by majority vote among the k most similar training files. It needs no training but must store and search all training files at prediction time, and it is sensitive to feature scaling.

Decision trees split the data recursively on the feature and threshold that best reduce impurity, giving rules that an analyst can read, but a single deep tree overfits.

Random Forest [3] grows many trees, each on a bootstrap sample of the files and considering a random subset of features at each split, and averages their votes; the averaging reduces variance. *Extra Trees* also chooses split thresholds at random, which adds diversity and speeds training.

Gradient boosting fits trees one after another, each to the errors that remain after the previous ones. XGBoost [9] adds a regularised objective and second order optimisation, and LightGBM [18] grows trees leaf by leaf on histograms of feature values, which makes it fast on large and sparse data.

Support vector machines find the boundary with the widest margin between the classes; with a radial basis kernel the boundary can be curved. The *linear support vector machine* used in Study 3 keeps a straight boundary, which suits very high dimensional sparse text features. *Multinomial naive Bayes* models term counts in the same setting.

The *multilayer perceptron* is a feed forward neural network with two hidden layers of rectified linear units, trained with the Adam optimiser and early stopping. It represents the neural approach at a scale that fits tabular header features.

Table 5 lists the settings. No hyperparameter search was run, so that every algorithm is compared at standard settings and no test data influence the choice.

Table 5. Algorithm settings used in Studies 1 to 3

Algorithm	Settings
Logistic regression	L2 penalty, C = 1 (Study 3: C = 10), standardised inputs
Gaussian naive Bayes	default variance smoothing, standardised inputs
Multinomial naive Bayes	additive smoothing 0.05 (Study 3)
k nearest neighbours	k = 5, Euclidean (Study 3: cosine)
Decision tree	unlimited depth, Gini impurity
Random Forest	300 trees (Study 2: 200, Study 3: 400), square root of features per split
Extra Trees	300 trees (Study 3: 400)
SVM	radial kernel, C = 10, Platt probabilities (Study 1)
Linear SVM	C = 0.5 (Study 3)
Multilayer perceptron	128 and 64 units, early stopping (Study 1)
XGBoost	400 rounds, depth 6, learning rate 0.08 (Study 1)
LightGBM	400 rounds (Study 3: 300), 63 leaves, learning rate 0.05

4.2 Study 1: static detection on ClaMP

The ClaMP integrated file holds 5,210 rows with 68 features and a label. The packer type column, a text field with 40 values such as NoPacker, was one hot encoded, giving 108 features. Three exact duplicates were removed, leaving 5,207 files (2,719 malware, 2,488 benign); no feature vector appeared with both labels. Ten algorithms were compared: logistic regression (the algorithm in the deployed engine), Gaussian naive Bayes, k nearest neighbours (k = 5), a decision tree, Random Forest and Extra Trees (300 trees each), a support vector machine with a radial kernel (C = 10), a multilayer perceptron with two hidden layers of 128 and 64 units, XGBoost and LightGBM (400 rounds each). Features were standardised for the linear, distance and neural models. We used stratified five fold cross validation and report the mean and 95 percent confidence interval of accuracy, together with precision, recall, F1, ROC AUC, false positive rate (FPR), false negative rate (FNR), training time and prediction time per file.

Table 6 groups the ClaMP features. Raw values come directly from the DOS header, the file header and the optional header; derived values describe section names, entropy, packing and the presence of version information.

Table 6. ClaMP feature groups

Group	Examples
DOS header	e_cblp, e_cp, e_lfanew
File header	number of sections, creation year, 15 Characteristics flags
Optional header	SizeOfCode, AddressOfEntryPoint, ImageBase, CheckSum, Subsystem, 11 DllCharacteristics flags, stack and heap sizes
Sections	counts of suspicious and normal section names
Entropy	entropy of the code section, data section and whole file
Packing	packer detected, packer type (40 values)
Other	file size, presence of version information

Two audits followed. First, the gain importance of a LightGBM model trained on all files ranked the features. Second, each feature on its own was used to train a depth three decision tree under the same cross validation; a feature that separates the classes well by itself may reflect how the dataset was collected rather than what malware is [2].

4.3 Study 2: detection over time on the Brazilian dataset

The Brazilian dataset contains benign PE files and malware files grouped by the day they were collected, from January 2013 to November 2020. We read every non empty daily file, removed repeated SHA1 hashes, dropped any hash that appeared as both benign and malware, and kept 20 numeric header fields (for example SizeOfCode, Entropy, NumberOfSections, DllCharacteristics), each transformed by (1+|x|). The lists of imported DLLs and imported functions were lowercased and hashed into 16,384 count features with a log(1 + x) transform. Benign files have no collection date, so they were split once at random into 60 percent for training, 20 percent for setting thresholds and 20 percent for testing, and the same benign files were used in every protocol.

Three protocols were compared with logistic regression, Random Forest (200 trees) and LightGBM: (a) a random split of all malware, 70 percent for training and 30 percent for testing, as in most published work; (b) a frozen model trained on 2013 malware only and tested on each later year; and (c) yearly retraining, where the model for year t is trained on all malware collected before year t. For every model the threshold that gives a 1 percent FPR on the threshold benign set was found, and we report detection (true positive rate) at that threshold on each test year, together with ROC AUC. Detection at a fixed FPR does not depend on the number of malware files in a test year, which varies from 7,207 in 2013 to 105 in 2020 (Table 9); F1 does, so it is reported only for the random split.

4.4 Study 3: family classification on Mal API 2019

Each Mal API 2019 sample is a sequence of Windows API call names recorded while the sample ran in a sandbox. Sequences range from 10 to 1.76 million calls (median 632). Consecutive repeats of the same call were collapsed and sequences were cut at 20,000 calls. We then removed 643 exact duplicates and 48 sequences that appeared with more than one family label, leaving 6,361 samples. Sequences were represented by TF IDF weights of API call unigrams, bigrams and trigrams (minimum document frequency 3, at most 150,000 terms, sublinear term frequency), fitted inside each training fold. Eight algorithms were compared under stratified five fold cross validation: logistic regression, a linear support vector machine, multinomial naive Bayes, k nearest neighbours with cosine distance, a decision tree, Random Forest and Extra Trees (400 trees each) and LightGBM (300 rounds), the last trained on the 5,000 terms with the highest chi square score in each training fold. We report accuracy and macro F1, the mean of the eight family F1 scores, which weights small families equally.

4.5 Study 4: mitigation policy

A detector produces a probability, not an action. We defined a three band policy. Files whose score is at or above a quarantine threshold t_q are quarantined automatically. Files with a score between a review threshold t_r and t_q are held for an analyst. Files below t_r are allowed. Both thresholds are set from benign files only: t_q is the score exceeded by 1 percent of benign files and t_r the score exceeded by 10 percent. Because t_q is fixed at a 1 percent benign FPR, the share of malware quarantined automatically is the detection rate at 1 percent FPR measured in Studies 1 and 2, for the random split, the frozen model and the yearly retrained model. We combine these results with a family based response playbook for files that are quarantined or confirmed by an analyst.

4.6 Evaluation measures

Let TP, FP, TN and FN be the numbers of true positives (malware flagged), false positives (benign files flagged), true negatives and false negatives (malware missed). Accuracy is (TP+TN)/(TP+FP+TN+FN); precision is TP/(TP+FP), the share of flagged files that are truly malicious; recall, also called the detection rate or true positive rate, is TP/(TP+FN); and F1 is the harmonic mean of precision and recall. The false positive rate is FP/(FP+TN) and the false negative rate is FN/(FN+TP). The ROC curve plots the true positive rate against the false positive rate as the decision threshold moves, and ROC AUC is the area under it, the probability that a randomly chosen malware file scores higher than a randomly chosen benign file. For family classification, the F1 score is computed for each family and averaged without weights (macro F1), so that small families such as adware count as much as large ones.

These measures answer different questions. Accuracy and F1 depend on the ratio of malware to benign files in the test set, which in practice is unknown and changes from day to day. The false positive rate decides how many legitimate programs an organisation will block, and the detection rate at a fixed false positive rate decides how much malware gets through at an acceptable level of disruption. For this reason the main operational measure in this paper is the detection rate at a 1 percent false positive rate, with the threshold chosen on benign files that are not used for testing.

5 Results

5.1 Static detection on ClaMP

Table 7 lists the ten algorithms ranked by F1. The three tree ensembles were statistically tied at the top: Random Forest reached 99.19 percent accuracy (95 percent confidence interval 98.99 to 99.39), LightGBM 99.15 and XGBoost 99.12, each with ROC AUC of 0.999 and an FPR near 1.1 to 1.2 percent. Extra Trees followed at 98.85 percent. The multilayer perceptron (97.95), decision tree (97.64) and support vector machine (97.50) formed a middle group, followed by k nearest neighbours (96.83) and logistic regression (96.04). Gaussian naive Bayes reached only 66.39 percent: it rarely raised false alarms (FPR 0.7 percent) but missed 64 percent of malware, because its assumption of independent, normally distributed features does not hold for header fields that are counts, flags and sizes spanning several orders of magnitude.

Table 7. Static malware detection on ClaMP, stratified five fold cross validation (5,207 files). Accuracy with 95 percent confidence interval; time is mean training time per fold and prediction time per file.

Algorithm	Accuracy (95% CI)	Precision	Recall	F1	ROC AUC	FPR	FNR	Train (s)	Predict (ms)
Random Forest	99.19 (98.99 to 99.39)	0.989	0.996	0.992	0.9994	1.2%	0.4%	0.68	0.058
LightGBM	99.15 (98.76 to 99.54)	0.990	0.994	0.992	0.9995	1.1%	0.6%	0.65	0.017
XGBoost	99.12 (98.67 to 99.57)	0.990	0.993	0.992	0.9995	1.1%	0.7%	0.43	0.004
Extra Trees	98.85 (98.45 to 99.24)	0.989	0.989	0.989	0.9991	1.2%	1.1%	0.59	0.064
Multilayer Perceptron	97.95 (97.26 to 98.63)	0.980	0.981	0.980	0.9942	2.2%	1.9%	0.74	0.003
Decision Tree	97.64 (97.22 to 98.06)	0.977	0.978	0.977	0.9763	2.5%	2.2%	0.04	0.001
Support Vector Machine	97.50 (96.67 to 98.34)	0.972	0.981	0.976	0.9962	3.1%	1.9%	1.06	0.043
K Nearest Neighbours	96.83 (96.30 to 97.37)	0.966	0.974	0.970	0.9910	3.7%	2.6%	0.01	0.023
Logistic Regression	96.04 (95.34 to 96.75)	0.961	0.964	0.962	0.9912	4.3%	3.6%	0.04	0.002
Gaussian Naive Bayes	66.39 (63.91 to 68.87)	0.984	0.363	0.529	0.9714	0.7%	63.7%	0.01	0.002

Logistic regression, the algorithm in CyberGuard's engine, made 4.3 percent false positives and missed 3.6 percent of malware, about three times the error of the ensembles on both counts. Its ROC AUC of 0.991 shows that it ranks files well, but a straight decision boundary cannot follow the interactions between header fields that the trees exploit. All models scored a file in well under a millisecond, so speed does not favour the linear model at this scale. XGBoost was the fastest of the leading group to predict (0.004 ms per file) and LightGBM the fastest to train among the large ensembles.

Figure 9 shows the ROC curves of the six best models, built from out of fold scores, with the false positive axis on a logarithmic scale below 1 percent. The ensembles reach a true positive rate above 0.9 while the FPR is still well below 1 percent; the curves of the support vector machine and the perceptron rise later. In a network that scans thousands of benign files a day, this low FPR region is where a detector is actually operated.

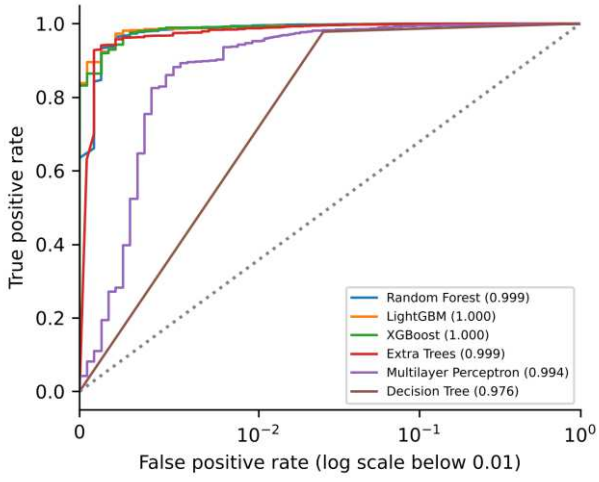

Figure 9. ROC curves on ClaMP for the six best algorithms (out of fold scores; AUC in brackets). The FPR axis is logarithmic below 0.01.

Figure 10 shows the features that LightGBM relied on. Two binary flags dominate: bit 12 of the file header Characteristics field takes 34.5 percent of the total gain and bit 2 of the optional header DllCharacteristics field 21.0 percent. File size (8.7 percent), the header CheckSum (6.1 percent), the Subsystem field and the stack reserve size follow.

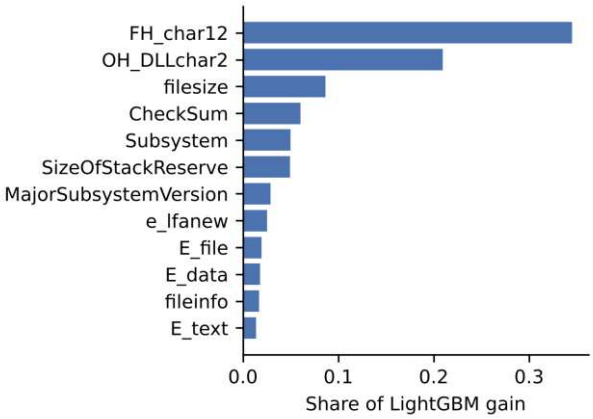

Figure 10. The twelve features with the largest share of LightGBM gain on ClaMP.

Table 8 gives the result of the single feature audit. The CheckSum field alone separated malware from benign files with a ROC AUC of 0.81, and the entropy of the whole file alone with 0.81. A correct header checksum is common in files built by Microsoft and other large vendors and rare in malware, so the checksum partly encodes where the benign files came from rather than whether a file is harmful. The ClaMP benign set was collected largely from Windows system folders; a detector trained on it learns, in part, *does this file look like a Windows system file*. Such shortcuts explain how simple models reach high accuracy on a single dataset and why the same models may fail on benign software from small vendors, which often lacks checksums and signatures. No feature alone came close to the ensembles' 0.999, however, so the models combine many weak signals.

Table 8. Single feature audit on ClaMP: ROC AUC of a depth three tree using one feature, five fold cross validation

Feature	ROC AUC alone
CheckSum	0.814
E file	0.807
sus sections	0.767
FH char12	0.767
MinorLinkerVersion	0.764
E data	0.760
fileinfo	0.751
E text	0.749

5.2 Detection over time on the Brazilian dataset

Table 9 shows the composition of the Brazilian data after cleaning: 21,095 benign files and 22,546 malware files, of which 7,207 were collected in 2013, 6,803 in 2014 and 5,426 in 2015, with far fewer in later years.

Table 9. Brazilian dataset after removing repeated and conflicting hashes

Group	Files
Benign (undated)	21,095
Malware 2013	7,207
Malware 2014	6,803
Malware 2015	5,426
Malware 2016	563
Malware 2017	1,867
Malware 2018	248
Malware 2019	327
Malware 2020	105
Malware total	22,546

Figure 11 shows the same distribution graphically. Most malware was collected in the first three years, and collection fell sharply after 2015. A random split therefore draws almost all of its test files from 2013 to 2015, the same period as most training files, while a time ordered evaluation tests on the scarcer and more recent files, which are the ones a deployed detector would face.

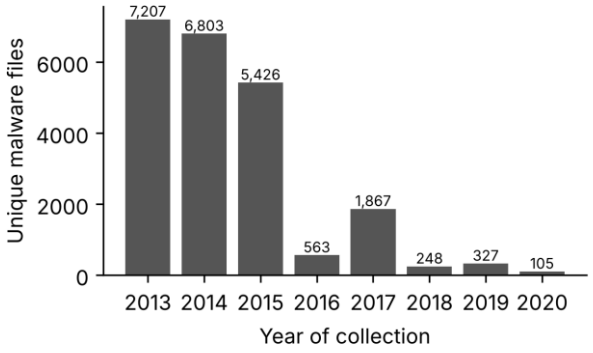

Figure 11. Unique malware files per year of collection in the Brazilian dataset after cleaning.

Table 10 compares the three protocols. Under a random split, every model looked excellent: LightGBM reached a ROC AUC of 0.998 and detected 97.8 percent of malware at a 1 percent FPR, Random Forest 96.3 percent and logistic regression 92.1 percent. When the model was trained on 2013 malware only, detection on 2014 malware was still close to the random split figure, but it then fell. Averaged over the test years 2015 to 2020, the frozen LightGBM model detected 76.3 percent of malware at the same FPR, Random Forest 74.1 percent and logistic regression 70.2 percent. By 2020 the frozen logistic regression model detected only 59.0 percent. Yearly retraining recovered much of the loss: averaged over 2015 to 2020, LightGBM detected 86.1 percent, Random Forest 81.8 percent and logistic regression 78.8 percent.

Table 10. Detection rate at 1 percent false positive rate on the Brazilian dataset under a random split, a model frozen after 2013, and yearly retraining. The random split mixes all years; ROC AUC in brackets for the random split.

Algorithm	Protocol	2014	2015	2016	2017	2018	2019	2020
LightGBM	random split			97.8 (AUC 0.998) on all years mixed				
	frozen after 2013	97.2	88.6	90.1	80.8	64.1	66.4	67.6
	retrained yearly	97.2	89.7	95.2	88.0	95.6	74.9	73.3
Random Forest	random split			96.3 (AUC 0.997) on all years mixed				
	frozen after 2013	96.6	87.1	92.5	77.3	68.5	62.1	57.1
	retrained yearly	96.6	87.9	92.5	80.9	91.5	67.3	70.5
Logistic Regression	random split			92.1 (AUC 0.989) on all years mixed				
	frozen after 2013	93.6	83.9	83.8	73.6	59.7	61.2	59.0
	retrained yearly	93.6	86.0	85.3	77.8	70.6	75.8	77.1

Figure 12 plots ROC AUC and detection at 1 percent FPR by test year for the frozen (dashed) and retrained (solid) models. The two panels tell different stories. ROC AUC of the tree models stays above 0.93 even for the frozen models, which would suggest that little has changed. Detection at a fixed FPR falls much further, because the new malware does not drop to the bottom of the ranking, it moves into the region where benign files also score. A threshold chosen for the old malware then lets more of it through. This is why reporting ROC AUC alone hides drift. The gap between the dashed and solid lines widens from 2017 onwards, the years in which the malware families seen in 2013 had been replaced. The frozen linear model degrades the most, which matters because CyberGuard's engine is a linear model.

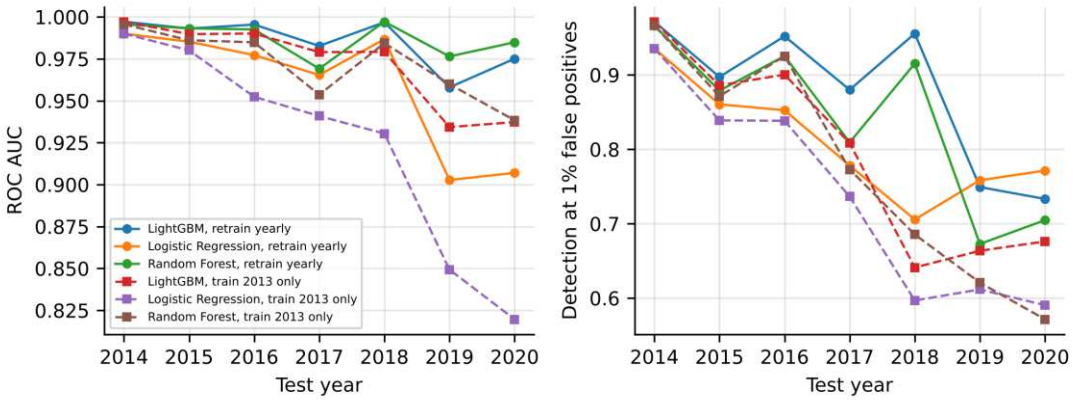

Figure 12. Detection over time on the Brazilian dataset. Left: ROC AUC by test year. Right: detection rate at 1 percent FPR. Dashed lines: model trained on 2013 malware only; solid lines: model retrained each year on all earlier malware.

The 2018 to 2020 points rest on 105 to 327 malware files each, so single year values carry wide uncertainty; a 95 percent binomial interval around 0.73 with 105 files spans roughly 0.64 to 0.81. The direction of the effect, however, is the same for all three algorithms and all later years.

5.3 Family classification on Mal API 2019

Collapsing repeated calls reduced the median sequence length from 632 to 296 calls. The audit found 643 exact duplicate samples and 48 sequences carrying two or more family labels, which no classifier can get right; both were removed. Table 11 ranks the eight algorithms. Family classification proved much harder than detection. LightGBM reached the highest scores, 65.5 percent accuracy and a macro F1 of 0.667, but logistic regression (65.0 percent, 0.662) and the linear support vector machine (64.7 percent, 0.655) were within one percentage point, and Extra Trees and Random Forest within two. Multinomial naive Bayes, the single decision tree and k nearest neighbours trailed by about ten points. With eight families, a classifier that guessed at random would be right about 12.5 percent of the time, so the API call patterns carry real family information, but about one sample in three is still assigned to the wrong family. Two reasons stand out. Family labels in Mal API 2019 come from antivirus engines, which often disagree on the family of the same file, and families such as trojans, droppers and downloaders share behaviour by design: each one loads, writes or fetches another program. The 48 sequences that appeared under two different labels are direct evidence of this label noise. The close scores of the linear models and the ensembles also show that, once API n grams are weighted by TF IDF, the classes are separated mainly along linear directions, so the extra capacity of the trees adds little.

Table 11. Eight family classification on Mal API 2019 (6,361 samples), stratified five fold cross validation, mean over folds.

Algorithm	Accuracy (%)	Macro F1
LightGBM	65.5	0.667
Logistic Regression	65.0	0.662
Linear SVM	64.7	0.655
Extra Trees	64.1	0.652
Random Forest	63.6	0.647
Multinomial Naive Bayes	56.1	0.561
Decision Tree	54.3	0.553
K Nearest Neighbours	54.7	0.552

5.4 Mitigation policy

Because the quarantine threshold is the score exceeded by 1 percent of benign files, the share of malware quarantined automatically equals the detection rate at 1 percent FPR reported in Table 10, and the share of benign files quarantined stays at about 1 percent in every year (measured benign FPR between 0.8 and 1.2 percent). Read this way, the drift results describe the policy directly. With a LightGBM model frozen after 2013, the share of each year’s malware quarantined automatically fell from 97.2 percent in 2014 to 64.1 percent in 2018 and 67.6 percent in 2020; the remainder fell into the review band or was allowed. With yearly retraining the share quarantined stayed at 88 to 97 percent until 2018 and was 73 to 75 percent in 2019 and 2020. For the frozen logistic regression model, the engine type deployed in CyberGuard, the quarantined share fell to 59.0 percent in 2020, so four in ten new malware files would depend on an analyst or would be released. On ClaMP, where training and test files come from the same period, the ensembles quarantined about 99 percent of malware at the same benign cost (Table 7). The review band, set at the score exceeded by 10 percent of benign files, adds a fixed analyst workload of about 9 percent of benign files plus the malware that falls between the two thresholds, and the growth of that malware share over time is the signal that the model needs retraining.

Table 12 gives the response attached to each family once a file is quarantined or confirmed by an analyst. It follows the containment, eradication and recovery stages of the NIST guidance [24, 32] and names the ATT&CK techniques [33] each family most often uses. The family label from Study 3 selects the row; when the family

classifier's confidence is low, the most cautious row that fits its top two families is applied. Because trojans, droppers and downloaders share behaviour and are the families most easily mistaken for one another, the playbook gives them the same containment step, isolation of the host, so a confusion among them does not weaken the response.

Table 12. Family based response playbook (CIA: the security property mainly at risk)

Family	CIA	Typical ATT&CK behaviour	Containment	Eradication and recovery
Adware	C	browser changes, unwanted downloads	quarantine file, block ad and download domains	remove bundled programs and browser extensions, reset browser
Backdoor	C, I	remote access, command and control	isolate host from network, block contacted addresses	rebuild host, rotate credentials used on it, review logins
Downloader	I	ingress tool transfer	isolate host, block download sources	search for second stage files by hash and path, remove scheduled tasks
Dropper	I	writes and launches payloads	isolate host	find and remove dropped files and persistence entries, rescan
Spyware	C	keylogging, screen and credential capture	isolate host, revoke sessions	reset passwords from a clean device, enable multifactor sign in
Trojan	C, I, A	varied, often loader or stealer	isolate host	full scan, remove persistence, restore changed files from backup
Virus	I	infects other executables	isolate host, block removable media	replace infected programs from trusted sources, rescan shares
Worm	A, I	lateral movement, exploitation	isolate host and segment, block spreading port	patch exploited service, scan all reachable hosts

6 Discussion

6.1 Which algorithm?

On static PE features the three tree ensembles were the most accurate, in line with Louk and Tama [20] and the boosted baseline of EMBER [1]. Their lead over logistic regression was large where it matters for operation: at the same 1 percent FPR on the Brazilian random split, LightGBM detected 97.8 percent of malware and logistic regression 92.1 percent, and on ClaMP the ensembles made about a third of the errors of the linear model. They also degraded more slowly over time. We therefore recommend that CyberGuard replace its logistic regression engine with LightGBM or XGBoost trained on EMBER features; both read the same 2,381 element vector, train in minutes and score a file in microseconds. For family classification the ordering was different: on API call sequences, LightGBM led by a narrow margin and logistic regression and the linear support vector machine were almost as accurate, much faster to train and easier to explain, because their weights point to the API calls behind each decision. At about 65 percent accuracy, a family label should guide the response, not decide it alone.

6.2 Why reported accuracy is not field accuracy

Three findings warn against reading a single accuracy figure, including the 100 percent shown on CyberGuard's dashboard (Figure 4), as a promise. First, the single feature audit on ClaMP showed that header checksums and entropy alone separate the classes well, so part of the accuracy comes from how the benign files were collected. Second, the random split on the Brazilian data overstated detection on future malware for every algorithm, by up to 33 percentage points for logistic regression in 2020. Third, Mal API 2019 contained duplicates and samples with conflicting labels, which inflate or cap scores depending on where they fall. These are the pitfalls described by Arp et al. [2] and Pendlebury et al. [26]. A dashboard should report accuracy on a time ordered test set, at the operating threshold, with the date range of the data.

6.3 Operating the detector

Retraining is the most effective defence against drift that we measured. Yearly retraining raised mean detection at 1 percent FPR from 2015 to 2020 by about 10 percentage points for LightGBM and 9 for logistic regression, and it costs only the effort of labelling new samples, which can come from files confirmed by analysts in the review band. More frequent retraining, monthly for example, should recover more [7, 10]. Thresholds must be reset at each retraining from fresh benign files, because the score distribution moves with the model.

The three band policy turns the score into actions whose cost can be planned. A fixed 1 percent of benign files is quarantined, which in an organisation that scans 1,000 new benign files a week means about ten files a week to release after checking. The review band adds a predictable analyst workload, and the share of malware that reaches it grows as the model ages, which makes the review band a drift alarm: when the share of files sent for review rises, the model is due for retraining [17].

6.4 Static and dynamic features in a web service

Studies 1 and 3 used different kinds of evidence, and the contrast is instructive. Static header features separated malware from benign files almost perfectly within one dataset, but they say little about what a program will do, so they support detection better than family classification. API call sequences describe behaviour, and they assigned eight families with about 65 percent accuracy from sequences alone, but collecting them requires running each file in a sandbox, which CyberGuard deliberately avoids. A practical design keeps static detection in the web service, where it is fast and safe, and sends only the files in the review band to an isolated sandbox whose API trace feeds a family classifier. This uses the expensive dynamic analysis only where it changes a decision, and it gives analysts behavioural evidence for exactly the files they must judge.

6.5 Relevance to Nigerian organisations

Many Nigerian universities, hospitals and small businesses have no security operations team and rely on

free or bundled antivirus products. A web based static analyser such as CyberGuard can give such organisations a second opinion on suspicious attachments and downloads without installing anything on their machines. Our results suggest two conditions for this to be useful. The model must be retrained on recent samples, because a model trained once and left alone loses a quarter or more of its detection within a few years, and the system must state its false positive rate honestly, because a tool that blocks legitimate software will be switched off. Sharing confirmed samples among institutions, for example through a national computer emergency response team, would provide the steady flow of labelled files that retraining requires.

6.6 Recommendations for CyberGuard

We recommend six changes. (1) Replace the logistic regression engine with a gradient boosted model and record the training data, date range and split. (2) Report accuracy, FPR and detection at 1 percent FPR on a time ordered test set on the dashboard, instead of a single test accuracy. (3) Replace the 0.5 threshold and the user's quarantine checkbox with the three band policy, with thresholds set from benign files. (4) Retrain at least yearly, preferably monthly, with analyst confirmed files. (5) Replace the heuristic category mapping with a family classifier; because CyberGuard does not execute files, this needs either a static family model trained on EMBER features with family labels [36] or an optional sandbox to record API calls. (6) Show the family playbook of Table 12 with each verdict and give clear error messages that separate an unreachable server from an invalid file (Figure 8).

6.7 Limitations

The three datasets are public research collections, not files seen by Nigerian organisations, and their benign sets are small and drawn from limited sources. ClaMP and the Brazilian data use header features close to, but not identical with, the EMBER vector used by CyberGuard; we could not retrain on EMBER itself because the computing environment could not download it, so our recommendation to change algorithm rests on the consistent ranking across datasets rather than on a test of the deployed vector. Benign files in the Brazilian data carry no dates, so the time ordered protocol ages the malware but not the benign software. Later years contain few malware files. Mal API 2019 labels come from antivirus engines and carry their disagreements. We did not test adversarial evasion, in which an attacker modifies a file to lower its score. Finally, the mitigation playbook is drawn from published guidance and was not tested in a live incident.

6.8 Ethics and safety

No malware was executed or stored in this study. All experiments used feature files and API traces published by the dataset authors. CyberGuard itself never executes uploaded files and locks quarantined files to read only.

7 Conclusion

This paper presented CyberGuard, a web based system that detects malware in Windows PE files by static analysis, assigns a category and recommends a response without ever running the file, and evaluated the methods it depends on with three public datasets. On ClaMP header features, Random Forest, LightGBM and XGBoost detected malware with about 99.2 percent accuracy and 1.1 to 1.2 percent false positives, against 96.0 percent accuracy and 4.3 percent false positives for logistic regression, the algorithm in the deployed engine. An audit showed that part of this accuracy comes from shortcuts such as the header checksum. On the Brazilian dataset, random splits overstated future performance for every algorithm: a model frozen after 2013 lost much of its detection by 2020, and yearly retraining recovered most of it. On Mal API 2019, eight algorithms assigned malware to eight families from API call sequences with at most 65.5 percent accuracy (LightGBM, macro F1 0.667), with linear models close behind, so family labels should guide rather than decide the response. A three band policy with thresholds set from benign files made the response predictable: a fixed 1 percent of benign files is quarantined, and the share of malware quarantined automatically falls as the model ages unless it is retrained. Future work should retrain the engine on EMBER and BODMAS with a time ordered split, add a static family classifier, retrain monthly from analyst confirmed files, and test the system against adversarial evasion and on files collected from Nigerian organisations.

Data and Code Availability

ClaMP, the Brazilian malware dataset and Mal API 2019 are publicly available from their authors. The experiment script that produces every table and figure in Section 5, with a fixed random seed, and the CyberGuard source code are available from the authors.

References

[1] Anderson, Hyrum S. and Roth, Phil. 2018. EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models. arXiv preprint arXiv:1804.04637.

[2] Arp, Daniel, Quiring, Erwin, Pendlebury, Feargus, Warnecke, Alexander, Pierazzi, Fabio, Wressnegger, Christian, Cavallaro, Lorenzo, and Rieck, Konrad. 2022. Dos and Don'ts of Machine Learning in Computer Security. In *Proceedings of the 31st USENIX Security Symposium*, 3971–3988.

[3] Breiman, Leo. 2001. Random Forests. *Machine Learning* 45, 1 (2001), 5–32. doi:10.1023/A:1010933404324

[4] Brown, Austin, Gupta, Maanak, and Abdelsalam, Mahmoud. 2024. Automated Machine Learning for Deep Learning Based Malware Detection. *Computers & Security* 137 (2024), 103582. doi:10.1016/j.cose.2023.103582

[5] Catak, Ferhat Ozgur, Ahmed, Javed, Sahinbas, Kevser, and Khand, Zahid Hussain. 2021. Data Augmentation Based Malware Detection Using Convolutional Neural Networks. *PeerJ Computer Science* 7 (2021), e346. doi:10.7717/peerj-cs.346

[6] Catak, Ferhat Ozgur and Yaz\i, Ahmet Faruk. 2019. A Benchmark API Call Dataset for Windows PE Malware Classification. arXiv preprint arXiv:1905.01999.

[7] Ceschin, Fabr\'\icio, Botacin, Marcus, Gomes, Heitor Murilo, Pinag\'e, Felipe, Oliveira, Luiz S., and Gr\'egio, Andr\'e. 2023. Fast & Furious: On the Modelling of Malware Detection as an Evolving Data Stream. *Expert Systems with Applications* 212 (2023), 118590. doi:10.1016/j.eswa.2022.118590

[8] Ceschin, Fabr\'\icio, Pinag\'e, Felipe, Castilho, Marcos, Menotti, David, Oliveira, Luiz S., and Gr\'egio, Andr\'e. 2018. The Need for Speed: An Analysis of Brazilian Malware Classifiers. *IEEE Security & Privacy* 16, 6 (2018), 31–41. doi:10.1109/MSEC.2018.2875369

[9] Chen, Tianqi and Guestrin, Carlos. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 785–794. doi:10.1145/2939672.2939785

[10] Chen, Yizheng, Ding, Zhoujie, and Wagner, David. 2023. Continuous Learning for Android Malware Detection. In *Proceedings of the 32nd USENIX Security Symposium*, 1127–1144.

[11] Dabas, Neeraj, Ahlawat, Prashant, and Sharma, Prabha. 2023. An Effective Malware Detection Method Using Hybrid Feature Selection and Machine Learning Algorithms. *Arabian Journal for Science and Engineering* 48 (2023), 9749–9767. doi:10.1007/s13369-022-07309-z

[12] Damodaran, Anusha, Di Troia, Fabio, Visaggio, Corrado Aaron, Austin, Thomas H., and Stamp, Mark. 2017. A Comparison of Static, Dynamic, and Hybrid Analysis for Malware Detection. *Journal of Computer Virology and Hacking Techniques* 13, 1 (2017), 1–12. doi:10.1007/s11416-015-0261-z

[13] Deldar, Fatemeh and Abadi, Mahdi. 2023. Deep Learning for Zero Day Malware Detection and Classification: A Survey. *ACM Computing Surveys* 56, 2 (2023), 1–37. doi:10.1145/3605775

[14] Galli, Antonio, La Gatta, Valerio, Moscato, Vincenzo, Postiglione, Marco, and Sperl\`\i, Giancarlo. 2024. Explainability in AI Based Behavioral Malware Detection Systems. *Computers & Security* 141 (2024), 103842. doi:10.1016/j.cose.2024.103842

[15] Gopinath, Mohana and Sethuraman, Sibi Chakkaravarthy. 2023. A Comprehensive Survey on Deep Learning Based Malware Detection Techniques. *Computer Science Review* 47 (2023), 100529. doi:10.1016/j.cosrev.2022.100529

[16] Hossain, Md. Alamgir and Islam, Md. Saiful. 2023. A Novel Hybrid Feature Selection and Ensemble Based Machine Learning Approach for Botnet Detection. *Scientific Reports* 13 (2023), 21207. doi:10.1038/s41598-023-48230-1

[17] Jordaney, Roberto, Sharad, Kumar, Dash, Santanu K., Wang, Zhi, Papini, Davide, Nouretdinov, Ilia, and Cavallaro, Lorenzo. 2017. Transcend: Detecting Concept Drift in Malware Classification Models. In *Proceedings of the 26th USENIX Security Symposium*, 625–642.

[18] Ke, Guolin, Meng, Qi, Finley, Thomas, Wang, Taifeng, Chen, Wei, Ma, Weidong, Ye, Qiwei, and Liu, Tie-Yan. 2017. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In *Advances in Neural Information Processing Systems 30*, 3146–3154.

[19] Kumar, Ajit, Kuppusamy, K. S., and Aghila, G.. 2019. A Learning Model to Detect Maliciousness of Portable Executable Using Integrated Feature Set. *Journal of King Saud University: Computer and Information Sciences* 31, 2 (2019), 252–265. doi:10.1016/j.jksuci.2017.01.003

[20] Louk, Maya Hilda Lestari and Tama, Bayu Adhi. 2022. Tree Based Classifier Ensembles for PE Malware Analysis: A Performance Revisit. *Algorithms* 15, 9 (2022), 332. doi:10.3390/a15090332

[21] Manzano, Cristian, Meneses, Claudio, Leger, Paul, and Fukuda, Hiroaki. 2022. An Empirical Evaluation of Supervised Learning Methods for Network Malware Identification Based on Feature Selection. *Complexity* 2022 (2022), 6760920. doi:10.1155/2022/6760920

[22] Nataraj, Lakshmanan, Karthikeyan, S., Jacob, Gregoire, and Manjunath, B. S.. 2011. Malware Images: Visualization and Automatic Classification. In *Proceedings of the 8th International Symposium on Visualization for Cyber Security (VizSec)*, 4:1–4:7. doi:10.1145/2016904.2016908

[23] Nazir, Ahsan and others. 2023. Advancing IoT Security: A Systematic Review of Machine Learning Approaches for the Detection of IoT Botnets. *Journal of King Saud University: Computer and Information Sciences* 35, 10 (2023), 101820. doi:10.1016/j.jksuci.2023.101820

[24] Nelson, Alex, Rekhi, Sanjay, Souppaya, Murugiah, and Scarfone, Karen. 2025. Incident Response Recommendations and Considerations for Cybersecurity Risk Management: A CSF 2.0 Community Profile. doi:10.6028/NIST.SP.800-61r3

[25] Pedregosa, Fabian and others. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[26] Pendlebury, Feargus, Pierazzi, Fabio, Jordaney, Roberto, Kinder, Johannes, and Cavallaro, Lorenzo. 2019. TESSERACT: Eliminating Experimental Bias in Malware Classification across Space and Time. In *Proceedings of the 28th USENIX Security Symposium*, 729–746.

[27] Peppes, Nikolaos, Alexakis, Theodoros, Adamopoulou, Evgenia, and Demestichas, Konstantinos. 2023. The Effectiveness of Zero Day Attacks Data Samples Generated via GANs on Deep Learning Classifiers. *Sensors* 23, 2 (2023), 900. doi:10.3390/s23020900

[28] Raff, Edward, Barker, Jon, Sylvester, Jared, Brandon, Robert, Catanzaro, Bryan, and Nicholas, Charles. 2017. Malware Detection by Eating a Whole EXE. arXiv preprint arXiv:1710.09435.

[29] Ronen, Royi, Radu, Marian, Feuerstein, Corina, Yom-Tov, Elad, and Ahmadi, Mansour. 2018. Microsoft Malware Classification Challenge. arXiv preprint arXiv:1802.10135.

[30] Roy, Kowshik Sankar, Ahmed, Tanim, Udas, Pritom Biswas, Karim, Md. Ebtidaul, and Majumdar, Sourav. 2023. A Hybrid Stacked Ensemble Learning Framework with Feature Engineering Schemes for Obfuscated Malware Analysis. *Intelligent Systems with Applications* 20 (2023), 200283. doi:10.1016/j.iswa.2023.200283

[31] Saxe, Joshua and Berlin, Konstantin. 2015. Deep Neural Network Based Malware Detection Using Two Dimensional Binary Program Features. In *2015 10th International Conference on Malicious and Unwanted Software (MALWARE)*, 11–20. doi:10.1109/MALWARE.2015.7413680

[32] Souppaya, Murugiah and Scarfone, Karen. 2013. Guide to Malware Incident Prevention and Handling for Desktops and Laptops. doi:10.6028/NIST.SP.800-83r1

[33] Strom, Blake E., Applebaum, Andy, Miller, Doug P., Nickels, Kathryn C., Pennington, Adam G., and Thomas, Cody B.. 2018. MITRE ATT&CK: Design and Philosophy.

[34] Vapnik, Vladimir N.. 1998. Statistical Learning Theory. Wiley, New York.

[35] Wolpert, David H. and Macready, William G.. 1997. No Free Lunch Theorems for Optimization. *IEEE Transactions on Evolutionary Computation* 1, 1 (1997), 67–82. doi:10.1109/4235.585893

[36] Yang, Limin, Ciptadi, Arridhana, Laziuk, Ihar, Ahmadzadeh, Ali, and Wang, Gang. 2021. BODMAS: An Open Dataset for Learning based Temporal Analysis of PE Malware. In *2021 IEEE Security and Privacy Workshops (SPW)*, 78–84. doi:10.1109/SPW53761.2021.00020