IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 11, November 2026

CyberGuard AI: A Two-Stage Machine Learning System for Network Intrusion Detection on CICIDS2017

S Davison[1], S Jebinraj[2], Harish K Bharath[3], R Adhithya[4], S Poornima[5]

Dept. of Computer Science and Engineering (Emerging Technologies),

SRM Institute of Science and Technology, Vadapalani Campus, Chennai, India[1-5]

Abstract: Signature-based intrusion detection systems detect only attacks for which a rule already exists, and many machine learning studies on the CICIDS2017 benchmark report optimistic results because of data leakage and accuracy-only evaluation. This paper presents CyberGuard AI, a web-based network intrusion detection system built on a two-stage supervised pipeline. A binary detector first separates benign flows from attacks, and an attack-family classifier then names the family of every flagged flow. Five candidate models built from four algorithms (Logistic Regression, Decision Tree, two Random Forest configurations and Histogram Gradient Boosting) compete at each stage and are selected on validation data only. Identifier columns are removed, preprocessing is fitted on training rows only, and near-duplicate flows are never divided between training and test data. Trained on a class-balanced sample of the original CICIDS2017 release, the selected Histogram Gradient Boosting detector reached 99.81% accuracy and an attack F1-score of 99.88% on an untouched test set of 7,234 flows, and the tuned Random Forest assigned all 5,749 test attacks to the correct family. The models are served through a FastAPI backend and a React dashboard that reports confidence, severity and plain-language explanations, and the complete system runs on a student laptop.

Keywords: intrusion detection, machine learning, CICIDS2017, network flow analysis, gradient boosting, random forest

I. INTRODUCTION

Organisations depend on continuous and secure network communication, and attacks such as denial of service (DoS), port scanning, brute-force logins, web exploits and botnets are hidden inside large volumes of legitimate traffic. Intrusion detection systems (IDS) monitor this traffic and raise alerts. Classic tools such as Snort [1] and Zeek (formerly Bro) [2] rely mainly on hand-written signatures, which are precise but detect only attacks for which a rule exists and require constant expert maintenance.

Supervised machine learning offers a complementary approach: a model learns the statistical patterns that separate malicious from benign traffic using labelled network flows described by features such as duration, packet counts, packet sizes, inter-arrival times and TCP flags. The CICIDS2017 dataset [3] has become a standard benchmark for this task. However, later work found flow-construction and labelling errors in the original release [4], [5], and many reported results are inflated by evaluation pitfalls such as random splits that place near-identical flows in both training and test sets, identifier features that let a model memorise the lab network, and accuracy-only reporting on imbalanced data [6].

This paper presents CyberGuard AI, a complete system that addresses these issues. Its contributions are: (1) a two-stage pipeline that separates the operational question "is this flow an attack?" from the finer question of the attack family; (2) a leakage-aware training protocol with header-based dataset-release detection, removal of identifier columns, training-only preprocessing and grouped or chronological splits; (3) a fair comparison of five candidate models selected on validation data only and evaluated once on an untouched test set with class-aware metrics; and (4) a deployable web application that reports confidence, severity and explanations for every flow.

II. RELATED WORK

Intrusion detection systems are commonly divided into signature-based systems, which match traffic against known attack patterns [1], [2], and anomaly-based systems, which model normal behaviour and flag deviations; Sommer and Paxson [7] showed that the latter often suffer from high false-alarm rates in real networks. Supervised machine learning sits between the two; Buczak and Guven [8] surveyed its use for intrusion detection and found that decision trees, ensemble methods, support vector machines and neural networks had all been applied successfully, with the choice of dataset and features strongly affecting the results.

 This work is licensed under a Creative Commons Attribution 4.0 International License

Older benchmarks such as KDD Cup 99 and NSL-KDD [9] are now considered outdated. CICIDS2017 [3] provides five days of realistic lab traffic with brute-force, DoS, DDoS, Heartbleed, web, infiltration, botnet and port-scan attacks, summarised by about 80 flow features. Engelen et al. [4] showed that the original flow extraction split some connections incorrectly and mislabelled many flows, and released a corrected version that adds an "Attempted" label for attack-session flows without a malicious payload; Lanvin et al. [5] independently reported labelling errors. Random Forests [10] and gradient boosting [11], [12] are widely used, strong classifiers for such tabular flow data. Arp et al. [6] catalogued pitfalls of machine learning in security research, including data snooping, spurious correlations and inappropriate performance measures, and Saito and Rehmsmeier [13] showed that precision-recall analysis is more informative than ROC analysis when the positive class is rare. CyberGuard AI builds on these findings by combining a leakage-aware protocol with class-aware metrics in a deployable system.

III. DATASET AND PREPROCESSING

A. Dataset Releases

The system supports the corrected 2021 release of CICIDS2017 (2,100,814 flows in five daily files) [4] and both packagings of the original release (2,830,743 flows in eight files) [3]. The release of every CSV file is detected automatically from its normalised column header, and files from different releases are never mixed. The experiments in this paper used the original MachineLearningCSV release, which contains 15 raw labels with extreme imbalance: about 80% of flows are benign, while Infiltration and Heartbleed have only 36 and 11 flows. The raw labels are grouped into ten families: BENIGN, DoS, DDoS, PortScan, BruteForce (FTP-Patator and SSH-Patator), WebAttack, Bot, Infiltration, Heartbleed and a safety-net OtherAttack class.

B. Cleaning and Feature Handling

Files are read in chunks of 50,000 rows. Column names are converted to lowercase identifiers, every feature is converted to a 32-bit float, and text, infinite and out-of-range values become missing values. Rows with no usable values, exact duplicates and identical feature vectors with conflicting labels are removed. Flow identifiers, IP addresses, the source port and the timestamp are excluded from the model inputs so that the model cannot memorise the lab network. After the split, features that are constant in the training rows are dropped and missing values are filled with training-set medians; Logistic Regression additionally uses a training-fitted standard scaler.

C. Leakage-Aware Splitting

The data are divided into approximately 70% training, 15% validation and 15% test. For the corrected release, flows are ordered by timestamp within each file and family, so that the model is tested on later traffic. The original release has no reliable timestamps, so every feature vector is rounded to four significant digits and rows with the same rounded vector form a group; within each family, whole groups are shuffled with a fixed seed of 42 and assigned to the partitions, so near-identical flows never appear in both training and test. In the default development mode, all files are scanned but at most 10,000 records per family are kept by seeded random priority, which preserves rare families and keeps training fast on a laptop.

IV. METHODOLOGY

A. Two-Stage Detection

Stage 1 is a binary detector that classifies every flow as BENIGN or ATTACK and outputs the attack probability. Stage 2 is an attack-family classifier trained only on attack flows and applied only to flows that Stage 1 flags as attacks, as shown in Fig. 1. A severity level is derived from the binary confidence: flows predicted as attacks are rated Low below 0.70, Medium from 0.70, High from 0.85 and Critical from 0.95, and benign flows are Informational.

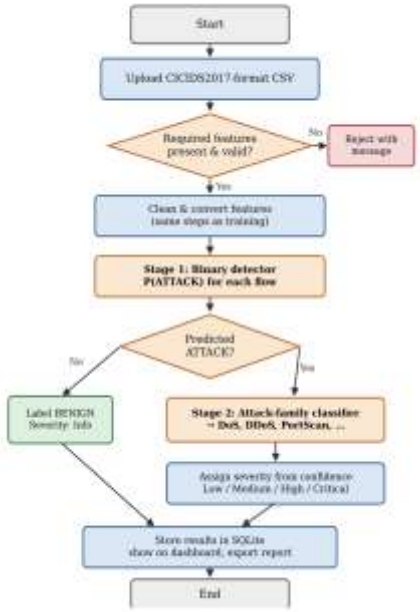


Fig. 1. Two-stage detection flow of CyberGuard AI.

Fig. 2.

B. *Candidate Models*

Five candidate models built from four scikit-learn [14] algorithms compete at each stage (Table I). All use class weighting to counter class imbalance. Logistic Regression models the attack probability as

$$P(y = 1|x) = \sigma(w^{\mathrm{T}}x + b) = \frac{1}{1 + e^{-(w^{\mathrm{T}}x + b)}} \qquad (1)$$

where x is the vector of scaled flow features and σ is the sigmoid function. Decision trees choose splits that minimise the Gini impurity

$$\mathrm{Gini}(S) = 1 - \sum_{k=1}^{K} p_k^2 \qquad (2)$$

where p_k is the proportion of class k among the K classes at a node. Random Forests [10] average the probabilities of many trees grown on bootstrap samples with random feature subsets. Gradient boosting [11] adds trees that fit the negative gradient of the loss, scaled by a learning rate η:

$$F_m(x) = F_{m-1}(x) + \eta\, h_m(x) \qquad (3)$$

The histogram variant, similar to LightGBM [12], bins each feature into at most 255 bins and uses early stopping on 10% of the training rows.

TABLE I. CANDIDATE MODELS AND SETTINGS

Model	Key settings
Logistic Regression	balanced class weights, max_iter = 2000, standard scaling
Decision Tree	max_depth = 20, min_samples_leaf = 3, balanced
Random Forest	200 trees, max_depth = 20, min_samples_leaf = 2, balanced
Random Forest (tuned)	200 trees, max_depth = 30, min_samples_leaf = 4, min_samples_split = 8, max_features = 0.7, balanced_subsample
Hist. Gradient Boosting	max_iter = 300, learning rate 0.1, 63 leaves, L2 = 1.0, balanced, early stopping

C. Selection and Evaluation

The binary detector is selected by the validation F1-score of the ATTACK class and the family classifier by the validation macro F1-score; ties keep the earlier, simpler candidate. Only the two winners are evaluated once on the test set, so the test set never influences model selection. With TP, FP and FN denoting true positives, false positives and false negatives,

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{4}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{5}$$

$$F_1 = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{6}$$

and the macro F1-score is the unweighted mean of the per-class F1-scores. ROC-AUC and PR-AUC (computed as average precision) are also reported, and the full two-stage cascade is evaluated end to end so that attacks missed by Stage 1 count as errors.

V. SYSTEM IMPLEMENTATION

CyberGuard AI follows a layered architecture (Fig. 2). A React dashboard communicates through a REST API with a FastAPI backend, which validates uploaded CSV files, calls the prediction module and stores every analysis and its per-flow results in an SQLite database. The machine learning layer applies exactly the same preprocessing at training and prediction time through saved scikit-learn pipelines, and every training run is stored in its own folder with its models, metrics, confusion matrices, feature importances and a held-out demo file. The dashboard provides upload and detection, filterable per-flow results with severity and plain-language explanations, analytics charts, detection history with CSV export and a model performance page. A one-click launcher creates the Python environment, installs the dependencies, trains the models when needed, builds the frontend and starts the server.

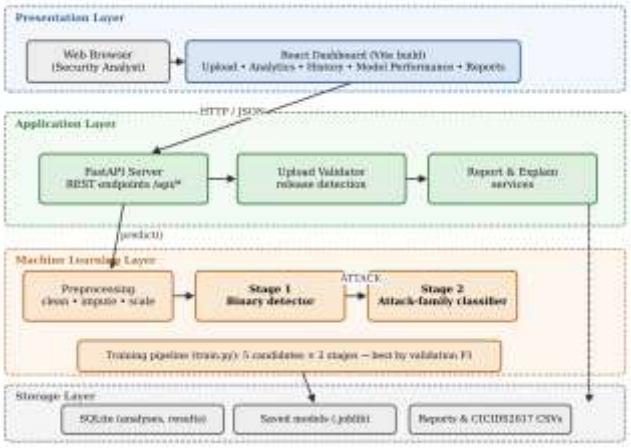


Fig. 3. Layered architecture of CyberGuard AI.

VI. EXPERIMENTAL RESULTS

A. Experimental Setup

The project's training script was run with its default settings on all eight MachineLearningCSV files on a Linux virtual machine with two CPU cores and about 4 GB of memory (Python 3.10.12, scikit-learn 1.7.2). Of the 2,830,743 flows read, development mode and cleaning produced 48,183 records (6,008 duplicates and 2 conflicting rows removed), split into 33,718 training, 7,231 validation and 7,234 test flows. The test set contained 1,485 benign flows and 5,749 attacks. Sixty-eight features were used after ten constant features were removed. The complete run took 113 s with a peak memory use of 0.44 GB. The project's 42 automated tests and its end-to-end demo script passed against the trained model.

B. Model Comparison

Table II lists the validation scores of all candidates. Tree-based models clearly outperformed Logistic Regression because the relationship between flow features and attacks is strongly non-linear. Histogram Gradient Boosting was

 This work is licensed under a Creative Commons Attribution 4.0 International License

selected for Stage 1 with an ATTACK F1-score of 0.9990, and the tuned Random Forest for Stage 2 with a macro F1-score of 0.9999.

TABLE II. VALIDATION F1-SCORES OF THE CANDIDATE MODELS

Model	Stage 1 ATTACK F1	Stage 2 macro F1
Logistic Regression	0.9717	0.9312
Decision Tree	0.9964	0.9573
Random Forest	0.9980	0.9988
Random Forest (tuned)	0.9980	0.9999
Hist. Gradient Boosting	0.9990	0.9994

C. Test-Set Results

Table III reports the test-set performance. The binary detector reached 99.81% accuracy, 99.90% attack precision and 99.86% attack recall. Its confusion matrix (Fig. 3) shows 8 missed attacks out of 5,749 and 6 false alarms out of 1,485 benign flows. The attack-family classifier assigned all 5,749 test attacks to the correct family, including the 5 Infiltration and 2 Heartbleed flows. In the end-to-end cascade, all 14 errors come from Stage 1; the lowest per-class recall was for Bot (0.9866), which accounted for 4 of the 8 missed attacks.

TABLE III. TEST-SET RESULTS OF THE SELECTED MODELS

Metric	Value
Binary accuracy	99.81%
Attack precision / recall	99.90% / 99.86%
Attack F1-score	99.88%
ROC-AUC / PR-AUC	0.9995 / 0.9998
Family classifier macro F1 (attack flows)	1.000
Cascade accuracy / macro F1	99.81% / 0.9979

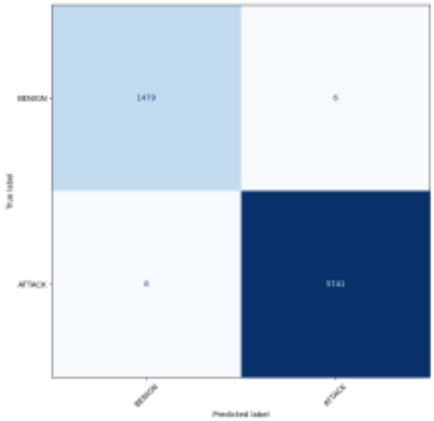


Fig. 4. Confusion matrix of the binary detector on the test set.

D. Feature Importance

Permutation importance identified destination_port (0.162), init_win_bytes_forward (0.113) and bwd_packet_length_min (0.075) as the most influential features of the binary detector. For the family classifier, the Random Forest impurity importance was highest for destination_port (0.315) and init_win_bytes_backward (0.205). These features reflect the services targeted by each attack in the CICIDS2017 lab, such as ports 21 and 22 for the brute-force attacks and port 80 for the DoS and DDoS attacks.

E. Application Output

The held-out demo file of 50 test flows, covering all nine classes, was analysed through the dashboard. All 50 flows were classified correctly, and the 43 attacks were reported as 42 Critical and 1 High severity alerts.

VII. DISCUSSION AND LIMITATIONS

The near-perfect family classification should be interpreted with care. The most important features partly encode how the CICIDS2017 lab was configured, so such separation is not expected to transfer unchanged to other networks. Development mode caps every family at 10,000 records, so benign flows formed about 20% of the test set rather than about 80% of the full dataset; at the real class balance, the same false-positive rate of 0.40% would yield lower precision. Infiltration and Heartbleed had only 5 and 2 test flows, so their scores are not statistically meaningful. Finally, the system is supervised and flow-based: it detects only known attack families, cannot inspect payloads, and analyses recorded CSV files rather than live traffic.

VIII.CONCLUSION AND FUTURE WORK

CyberGuard AI combines a two-stage detection pipeline, a leakage-aware training protocol and a usable web application. On an untouched test set from CICIDS2017, the selected Histogram Gradient Boosting detector reached an attack F1-score of 99.88%, and the tuned Random Forest classified every test attack into the correct family. Future work includes training on every record and on the corrected 2021 release with its chronological split, cross-dataset validation on UNSW-NB15 [15], live flow capture, unsupervised detection of unknown attacks, and per-prediction explanations using SHAP [16].

ACKNOWLEDGMENT

The authors thank G. Karthiha, Assistant Professor (O.G), and Dr. P. Chitra, Professor and Head, Department of Computer Science and Engineering (Emerging Technologies), SRM Institute of Science and Technology, Vadapalani Campus, for their guidance and support.

REFERENCES

[1] M. Roesch, "Snort – Lightweight intrusion detection for networks," in Proc. 13th USENIX Conf. System Administration (LISA '99), Seattle, WA, USA, 1999, pp. 229–238.
[2] V. Paxson, "Bro: A system for detecting network intruders in real-time," Computer Networks, vol. 31, no. 23–24, pp. 2435–2463, 1999.
[3] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in Proc. 4th Int. Conf. Information Systems Security and Privacy (ICISSP), Funchal, Portugal, 2018, pp. 108–116.
[4] G. Engelen, V. Rimmer, and W. Joosen, "Troubleshooting an intrusion detection dataset: the CICIDS2017 case study," in Proc. IEEE Security and Privacy Workshops (SPW), San Francisco, CA, USA, 2021, pp. 7–12.
[5] M. Lanvin, P.-F. Gimenez, Y. Han, F. Majorczyk, L. Mé, and É. Totel, "Errors in the CICIDS2017 dataset and the significant differences in detection performances it makes," in Proc. 17th Int. Conf. Risks and Security of Internet and Systems (CRiSIS 2022), LNCS, vol. 13857. Cham, Switzerland: Springer, 2023, pp. 18–33.
[6] D. Arp, E. Quiring, F. Pendlebury, A. Warnecke, F. Pierazzi, C. Wressnegger, L. Cavallaro, and K. Rieck, "Dos and don'ts of machine learning in computer security," in Proc. 31st USENIX Security Symp., Boston, MA, USA, 2022, pp. 3971–3988.
[7] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in Proc. IEEE Symp. Security and Privacy, Oakland, CA, USA, 2010, pp. 305–316.
[8] A. L. Buczak and E. Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection," IEEE Commun. Surveys Tuts., vol. 18, no. 2, pp. 1153–1176, 2016.
[9] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the KDD CUP 99 data set," in Proc. IEEE Symp. Computational Intelligence for Security and Defense Applications (CISDA), Ottawa, Canada, 2009, pp. 1–6.
[10] L. Breiman, "Random forests," Machine Learning, vol. 45, no. 1, pp. 5–32, 2001.
[11] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," The Annals of Statistics, vol. 29, no. 5, pp. 1189–1232, 2001.
[12] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "LightGBM: A highly efficient gradient boosting decision tree," in Advances in Neural Information Processing Systems 30 (NeurIPS), 2017, pp. 3146–3154.
[13] T. Saito and M. Rehmsmeier, "The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets," PLoS ONE, vol. 10, no. 3, e0118432, 2015.
[14] F. Pedregosa et al., "Scikit-learn: Machine learning in Python," J. Machine Learning Research, vol. 12, pp. 2825–2830, 2011.

 This work is licensed under a Creative Commons Attribution 4.0 International License

[15] N. Moustafa and J. Slay, "UNSW-NB15: A comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)," in Proc. Military Communications and Information Systems Conf. (MilCIS), Canberra, Australia, 2015, pp. 1–6.

[16] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in Advances in Neural Information Processing Systems 30 (NeurIPS), 2017, pp. 4765–4774.