



ML-Based Intrusion Detection System with Intrusion Prevention System

Prathysh N, Rajesh H S, Purushothama M, Chandrashekar Y

Department of Computer Science and Engineering, RV Institute of Technology and Management, Bengaluru, India

Under the guidance of Dr. Syeda Ayesha Unisa

Department of Computer Science and Engineering, RV Institute of Technology and Management, Bengaluru, India

How to Cite this Article:

N, P., S, R. H., M, P. & Y, C. (2026). ML-Based Intrusion Detection System with Intrusion Prevention System. International Journal of Creative and Open Research in Engineering and Management, 2(10), 1-9.
<https://doi.org/10.55041/ijcope.v2i10.112>

License:

This article is published under the terms of the Creative Commons Attribution 4.0 International License (CC BY 4.0), which permits unrestricted use, distribution, and reproduction in any medium, provided the original author(s) and the source are credited.

© The Author(s). Published by International Journal of Creative and Open Research in Engineering and Management.



<https://doi.org/10.55041/ijcope.v2i10.112>

Abstract—Phishing pages, malware droppers, ransomware payment portals, and botnet command-and-control (C2) domains are, in the overwhelming majority of cases, still reached through an ordinary click on an ordinary link. Network-level intrusion detection systems that rely on full packet capture are effective but heavy to deploy for a single user or a small lab, and public flow-based benchmarks describe attacks that a browser never actually sees. This paper presents a lightweight hybrid intrusion detection and prevention system that intercepts navigation at the browser itself, combining a fast, explainable rule engine (keyword matching, hostname-entropy scoring, and a threat-intelligence hook) with an ensemble of four supervised classifiers — Random Forest, Decision Tree, Support Vector Machine, and Gaussian Naive Bayes — trained on lexical and structural URL features. On a held-out test set of 72 URLs, Random Forest reached 95.83% accuracy and a 95.52% F1-score and was selected as the production model. We also report a concrete false-positive case study: an early version of the entropy rule, computed over the full URL rather than the hostname alone, blocked 7 of 10 completely ordinary sites (Gmail, Google Docs, Amazon, LinkedIn, and others) because their query strings and document IDs looked statistically random to the model. Restricting the entropy check to the hostname and rebalancing the training corpus so that benign URLs also contain realistic long identifiers brought this down to 1 false positive in 12, with no loss of true-positive detections. Finally, the system exposes a live, per-attack-type accuracy view that scores all four trained algorithms against every real request as it arrives, rather than only against a one-time offline split.

Index Terms—Intrusion Detection System, Intrusion Prevention System, phishing detection, Random Forest, domain generation algorithm, browser extension, malicious URL classification, machine learning.

somewhere else: inside the browser, at the moment a navigation is requested, before the page is ever fetched.

This paper describes exactly that kind of system, built and evaluated end to end rather than only simulated. Its contributions are threefold. First, a hybrid architecture is presented that intercepts every navigation from a Chrome extension, checks it against an explainable rule engine and a four-algorithm machine learning ensemble running behind a local REST API, and renders an actual block page with the reason for the block. Second, a false-positive failure is documented and fixed with numbers attached, rather than assumed away: an entropy-based rule that looked at the whole URL, and a training corpus that only ever put random-looking tokens in the malicious class, together caused 7 of 10 completely ordinary websites to be blocked; isolating and correcting both causes brought that down to 1 of 12 with no loss of true detections. Third, the system includes a live per-attack-type accuracy view that scores every trained algorithm against every real request as it happens, which is a smaller but, we think, practically useful complement to the usual one-time offline benchmark table.

The rest of the paper is organized as follows. Section II reviews related work in URL-based and DGA-based detection. Section III describes the system architecture.

I. INTRODUCTION

Most successful compromises of an ordinary user's machine do not begin with a buffer overflow or a cracked password; they begin with a link. A message arrives, a search result looks plausible, a shortened URL gets pasted into a chat window, and the browser is asked to load something that was never meant to be trusted. For an individual user, a student lab, or a small organization without a security operations team, the browser tab is the real network perimeter, whether or not anyone has provisioned one.

Two broad families of defense already exist for this problem, and both have a gap that matters at this scale. Static blacklists such as public phishing-report feeds are precise but reactive: a freshly registered malicious domain is, by definition, not yet on any list. Network-level intrusion detection systems trained on flow-based benchmarks are far more general, but they assume a vantage point — a spanned switch port, a network tap, a firewall that can mirror traffic — that a personal laptop simply does not have. A system that wants to protect one machine in real time has to sit



Section IV covers feature engineering, including the hostname-entropy feature that the case study in Section VI centers on. Section V describes the dataset and the detection pipeline. Section VI presents the offline algorithm comparison, the false-positive case study, and the live monitoring view. Section VII discusses limitations and future work, and Section VIII concludes.

II. RELATED WORK

A. URL-Level and Lexical Detection

Ma et al. [4] showed early on that a malicious website can often be distinguished from a benign one using only lexical and host-based features of its URL — length, token composition, registration and DNS properties — without ever fetching the page content, and that this can be done with online learning at a scale suited to real traffic. Sahingoz et al. [3] later demonstrated that lexical features alone, evaluated with several classical machine learning models on a large real-world phishing corpus, can reach high accuracy without depending on third-party services such as WHOIS or search-engine rank, which matters for a system that has to answer in milliseconds from a browser extension. Both results motivate the choice, made in this work, to rely on cheap string-level features computed locally rather than on network lookups.

Several surveys have since consolidated this line of work. Sahoo et al. [11] observe that blacklists cannot be exhaustive and cannot detect newly generated malicious URLs, which is the gap that learned detectors target, and Aljabri et al. [12] review the machine-learning techniques, feature types, and datasets used for malicious-URL detection together with the limitations of each. In an experimental study on 66,506 URL records, Aljabri et al. [13] evaluated lexical, network, and content-based features with machine-learning and deep-learning models and found Naive Bayes to be the best performer, whereas it is the lowest-accuracy model on our feature set (Section VI-A); the ranking of classifiers therefore depends on the dataset and the feature design. At the system level, Marchal et al. [14] describe PhishStorm, which analyzes URLs in real time and can sit behind an email server or HTTP proxy, a deployment point close in spirit to the browser-level interception used here.

B. Domain Generation Algorithm Detection

A closely related line of work targets domain generation algorithms (DGAs), where malware families periodically generate large numbers of pseudo-random hostnames to reach a command-and-control server. Li et al. [5] built a machine learning framework for DGA detection evaluated on a full year of real traffic, and Bhatia et al. [6] proposed an entropy- and likelihood-based detector built directly on the hypothesis that human-chosen domain names have measurably lower entropy than algorithmically generated ones. Yadav et al. [15] approached the same problem from the distribution side, showing that the character and bigram distributions of algorithmically generated domain names differ measurably from those of human-chosen names. This hypothesis is exactly what the hostname-entropy feature in Section IV relies on, and Section VI reports the entropy values actually

measured on both classes in this project rather than only citing the general claim.

C. Flow-Based Intrusion Detection and Algorithmic Background

At the network layer, the CICIDS2017 dataset of Sharafaldin et al. [1] remains a standard benchmark for flow-based intrusion detection, built from several days of realistic background traffic mixed with staged attacks and labeled with more than seventy flow-level features. Buczak and Guven [7] provide a broader survey of the data-mining and machine-learning methods applied to cyber intrusion detection generally, of which URL-level classification is one specific instance rather than a separate discipline. Khraisat et al. [16] classify intrusion detection systems into signature-based and anomaly-based systems and survey the datasets used to evaluate them, and Gümüşbaş et al. [17] review the databases and deep-learning methods used for the same purpose. Vinayakumar et al. [18] report that deep neural networks compare favorably with classical classifiers on several public benchmark datasets. CICIDS2017 is a natural reference point for any IDS project, and it was the original dataset of choice for this project as well. In practice, however, it describes traffic captured with CICFlowMeter at a network vantage point, which a browser extension does not have access to; adapting flow-level features to a single-endpoint, URL-triggered detector would have meant simulating a network tap that does not exist in this deployment. The system in this paper is therefore positioned as complementary to, rather than a reimplementing of, flow-based IDS work: it operates one layer higher, at the URL itself, and is meant to sit on a single machine rather than at a network chokepoint. The four classifiers compared in Section VI follow, respectively, the ensemble formulation of Breiman [2], the classification-and-regression-tree framework of Breiman et al. [9], and the maximum-margin formulation of Cortes and Vapnik [8].

D. Comparative Summary

Table IV summarizes where this work sits relative to the systems discussed above, along the two dimensions that matter most for a browser-deployed detector: the layer at which detection happens, and whether the reported system is designed to act in real time on a single navigation rather than only to classify a URL offline. Most of the cited work operates at the URL or DNS layer, consistent with the position taken in Section II-A and II-B, and none of it combines an explainable rule stage with a live, per-category monitoring view of the kind described in Section VI-C.

TABLE IV. QUALITATIVE COMPARISON WITH RELATED WORK

Ref.	Layer	Approach	Real-Time
[1]	Network flow	Labeled flow dataset (benchmark)	No
[3]	URL / lexical	Classical ML on lexical features	Yes
[4]	URL / host	Online learning, lexical+host	Yes
[5]	DNS / DGA	ML framework on real traffic	Yes
[6]	DNS / DGA	Entropy-likelihood detector	Yes
This work	Browser URL	Hybrid rules + 4-model ensemble	Yes (<6 ms)



III. SYSTEM ARCHITECTURE

A. Component Overview

The system has three cooperating parts, shown in Fig. 1. A Chrome extension, built on Manifest V3, registers a background service worker that listens for every outgoing navigation. Before a page is allowed to load, the candidate URL is sent as a POST request to a locally running Flask API. If the verdict is "blocked", the extension redirects the tab to a bundled block page rather than letting navigation continue; this redirect, not the API call by itself, is what makes the system a prevention system rather than only a detection system.

The Flask API is the decision engine. At startup it loads a persisted production model together with the four individually trained algorithms described in Section V, and exposes five endpoints: /check_url for the actual detection decision, /get_logs for the historical block list, /algorithm_accuracy for the one-time offline benchmark, /live_accuracy for the real-time per-category view described in Section VI-C, and /settings, which lets an operator adjust the hostname-entropy threshold at runtime without restarting the service.

The Streamlit dashboard is the analyst-facing layer. It polls the API on a short interval and presents a live threat feed, time-based analytics, the live and offline algorithm comparisons, and PDF/CSV export of the incident log, all behind a login screen.

B. Implementation Stack

The detection API is written in Python 3 using Flask, with scikit-learn [19] for all four classifiers and joblib for model persistence. The dashboard is built with Streamlit and Plotly for the interactive charts described in Section VI. The extension targets Chrome's Manifest V3 [10], which replaced the older persistent background page with an event-driven service worker; this has a practical consequence for a security extension, since the worker can be unloaded by the browser when idle and is re-instantiated on the next navigation event rather than kept permanently resident. Detection state — the incident log and the live per-algorithm accuracy log described in Section VI-C — is persisted to disk as JSON on the API side rather than kept only in memory, specifically so that a server restart during a demonstration or a normal development cycle does not silently discard prior detections.

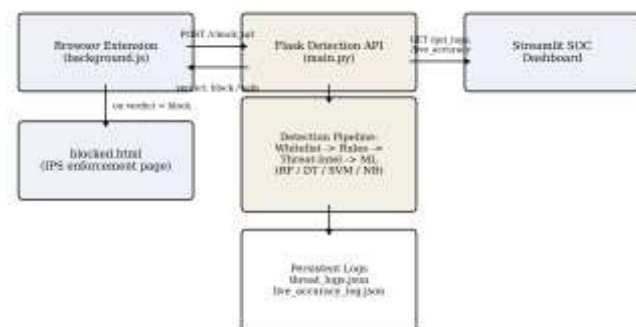


Fig. 1. System architecture: browser-side enforcement, a Flask decision engine, and a Streamlit analyst dashboard.

IV. FEATURE ENGINEERING

Every URL is reduced to eight features before being handed to a classifier: total URL length, digit count, a count of structural special characters (. @ ? & = _ -), the Shannon entropy of the full URL string, the Shannon entropy of the hostname alone, the combined length of the path and query string, and the number of subdomain levels, plus a binary flag for whether the hostname is a raw IP address. Shannon entropy [20] is computed in the usual way,

$$H(s) = - \sum p_i \log_2 p_i,$$

where p_i is the observed frequency of the i -th distinct character in the string s .

The hostname-only entropy feature was added specifically to correct the false-positive failure described in Section VI-B, and is the more important of the two entropy features in production. Table II reports the entropy actually measured on a small set of well-known benign hostnames next to a set of clearly DGA-style malicious hostnames generated for this project; the separation — roughly 2.9–3.6 for ordinary hostnames against 3.4–4.9 for algorithmically generated ones — is what motivates the 4.0 default threshold used by the rule engine in Section V.

TABLE II. HOSTNAME ENTROPY: BENIGN VS. MALICIOUS

Hostname	Domain Entropy
www.nytimes.com	3.32 (benign)
docs.google.com	2.97 (benign)
www.flipkart.com	3.58 (benign)
g00gle-secure.click	3.43 (malicious)
freegiftclaim-x7z9q.xyz	4.09 (malicious)
qwe8xzm9p0asdghjkl123456789.info	4.86 (malicious)

V. DATASET AND DETECTION PIPELINE

A. Synthetic URL Dataset

Because the system operates on URLs rather than network flows, a dedicated URL-level dataset was generated rather than adapted from CICIDS2017. The generator produces 359 unique URLs: 189 benign and 170 malicious. Benign URLs combine a list of well-known domains with either a plain path (e.g., /about, /login) or, for roughly 45% of benign samples, a realistic long identifier modeled on real services — a Google Docs document ID, an Amazon product reference, a LinkedIn activity ID, a GitHub diff hash, and similar patterns. This second group exists specifically to prevent the classifier from treating any long, random-looking token as inherently suspicious, which is exactly the shortcut Section VI-B shows the model taking when that group was absent. Malicious URLs are drawn from four templates: phishing- and malware-keyword patterns embedded in otherwise ordinary-looking paths, a ransomware-payment keyword pattern, and DGA-style hostnames generated as random alphanumeric strings under uncommon top-level domains. The dataset is split 80/20 with stratification on the label, giving 287 training and 72 test examples.



B. Detection Pipeline

Every request passes through the same ordered pipeline. A 13-domain static allowlist is checked first and returns "safe" immediately for well-known platforms, avoiding unnecessary computation on the highest-volume traffic. Keyword rules then check for the substrings "phish", "malware"/"trojan", and "ransomware" in the URL. If none match, the hostname-entropy rule fires when the hostname's Shannon entropy exceeds the configurable threshold (4.0 by default) and the URL is not a known shortener. A threat-intelligence lookup runs next as a hook for an external blacklist feed. Only if none of these explainable stages produce a verdict does the production Random Forest model get the final say, in which case the incident is logged as an "ML-Detected Anomaly" so that an analyst can immediately see, from the reason field alone, whether a given block was rule-driven or model-driven.

VI. RESULTS AND DISCUSSION

A. Offline Algorithm Comparison

Table I reports accuracy, precision, recall, F1-score, false-positive rate (FPR), and false-negative rate (FNR) for all four algorithms on the 72-sample held-out test set. Random Forest is the strongest across every balanced metric (95.83% accuracy, 95.52% F1) and is used as the production model. Fig. 2 plots accuracy, precision, recall, and F1-score for all four algorithms side by side, and Fig. 3 compares their false-positive and false-negative rates. Decision Tree and SVM sit in the middle, with SVM trading a slightly higher false-positive rate for a lower false-negative rate than Decision Tree. Naive Bayes is the most striking result in the table: perfect precision (100%) paired with very poor recall (14.71%), meaning it almost never raises a false alarm but also misses the great majority of actual malicious URLs. This is consistent with Naive Bayes' conditional-independence assumption breaking down on this feature set — url_length, digit_count, and special_chars are naturally correlated with one another, so the model ends up requiring an unusually extreme combination of feature values before it will commit to a positive prediction. Fig. 3 shows this trade-off directly: Naive Bayes has the lowest false-positive rate but by far the highest false-negative rate, while Random Forest is the only model that keeps both rates low.

TABLE I. OFFLINE ALGORITHM COMPARISON (72-SAMPLE TEST SET)

Algorithm	Acc%	Prec%	Rec%	F1%	FPR%	FNR%
Random Forest	95.83	96.97	94.12	95.52	2.63	5.88
Decision Tree	87.50	87.88	85.29	86.57	10.53	14.71
SVM	88.89	86.11	91.18	88.57	13.16	8.82
Naive Bayes	59.72	100.0	14.71	25.64	0.00	85.29

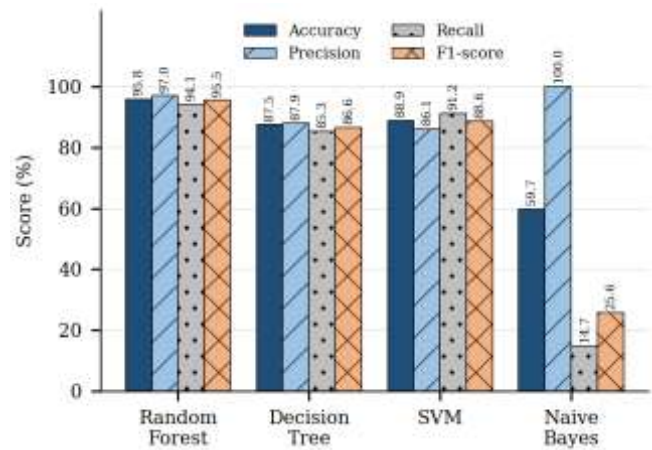


Fig. 2. Accuracy, precision, recall and F1-score of the four classifiers on the 72-sample test set.

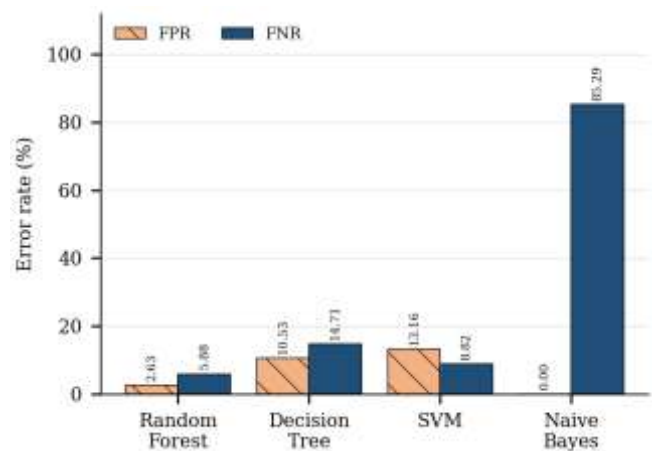


Fig. 3. False-positive and false-negative rates of the four classifiers on the 72-sample test set.

B. False-Positive Case Study

The offline numbers in Table I describe performance on synthetic held-out data; the more informative test is what the system does on real traffic. An early version of the pipeline computed entropy over the entire URL — path and query string included — and used a threshold of 4.5. Testing this version against ten completely ordinary destinations (Gmail, Google Docs, an Amazon product page, Wikipedia, a Reddit thread, Outlook, LinkedIn, ChatGPT, WhatsApp Web, and Flipkart) produced seven false blocks.

Diagnosis pointed to two independent causes that were both present and additive. First, real services routinely embed long, high-entropy identifiers in the path or query string of a completely legitimate URL — a document ID, a tracking parameter, a session token — so scoring the whole URL conflates ordinary web-application plumbing with an actual algorithmically generated hostname. Second, the original training corpus had only ever placed random-looking tokens in the malicious class; every classifier had, in effect, learned the rule "a long random substring anywhere in the URL means malicious," a correlation that happens to hold in a small synthetic dataset but does not hold on the real web.

Both causes were corrected together: the entropy rule was restricted to the hostname only (the domain_entropy feature introduced in Section IV), and the benign half of the training corpus was rebuilt so that roughly half of its samples also



carry realistic long identifiers, removing the spurious correlation. Repeating the same ten-site test, extended to twelve sites, after these two changes produced one false positive, with all six labeled malicious test URLs still correctly blocked. Table II, discussed in Section IV, gives the underlying hostname-entropy values that made the 4.0 threshold a reasonable choice once the feature was correctly scoped.

C. Live Per-Category Accuracy Monitoring

The numbers in Table I are computed once, at training time, against a fixed test split. In production, the system additionally scores all four trained algorithms — not only the deployed Random Forest model — against every single incoming request, and aggregates agreement with the final verdict by attack category: Phishing, Malware, Ransomware, DGA Botnet, ML-Detected Anomaly, and Benign. This is exposed on the dashboard as a live, auto-refreshing view rather than a static report, so a drift in any one algorithm's agreement rate on a specific attack category becomes visible immediately, without waiting for a scheduled retraining evaluation. This is a modest contribution next to the offline benchmark, but it is the piece that turns a one-time accuracy number into something closer to an always-on monitoring signal.

D. Confusion Matrix for the Production Model

Table III breaks the Random Forest result down into the four raw outcome counts on the 72-sample test set, rather than only the derived rates already given in Table I. Of the 34 samples actually labeled malicious, 32 were correctly flagged and 2 were missed; of the 38 samples actually labeled benign, 37 were correctly passed and 1 was wrongly blocked. The two missed detections are of more practical concern than the single false positive, since a missed malicious URL reaches the user while a false positive only costs an unnecessary block; both numbers are small enough, at this dataset size, that neither should be over-interpreted as a stable long-run error rate.

TABLE III. RANDOM FOREST CONFUSION MATRIX (72-SAMPLE TEST SET)

	Predicted Benign	Predicted Malicious
Actual Benign	37 (TN)	1 (FP)
Actual Malicious	2 (FN)	32 (TP)

E. Inference Latency

Because the extension blocks navigation until a verdict is returned, detection latency is a user-facing concern and not only an offline metric. Averaged over 200 repeated predictions on the same machine used for training, the production Random Forest model returns a prediction in 5.68 ms, Decision Tree in 0.60 ms, SVM in 0.67 ms, and Naive Bayes in 0.67 ms; feature extraction itself takes under 0.03 ms per URL. On the request path actually exercised by a user's browser, only the allowlist check, the rule engine, and the single production model run, so the added latency per navigation is on the order of a few milliseconds — negligible next to the time already spent on DNS resolution and the TLS handshake for the destination itself. The live monitoring view

in Section VI-C additionally runs all four models on every request purely for comparison logging, which is why it is implemented as a background scoring step rather than something the user's navigation waits on.

VII. LIMITATIONS AND FUTURE WORK

The dataset, at 359 synthetic samples, is a proof-of-concept scale rather than a substitute for training on real captured phishing and malware corpora such as PhishTank, OpenPhish, or URLhaus, and results should be read with that in mind. This caution is in line with Sommer and Paxson [21], who discuss why machine-learning detectors are difficult to evaluate and to deploy outside the setting they were trained in. The attack-type label attached to each detection (Phishing, Malware, and so on) is currently assigned by the rule layer, not learned directly by the classifiers, which are trained as binary malicious/benign detectors; a genuinely multi-class classifier is a natural next step. The threat-intelligence check is currently a placeholder hook rather than a live call to a service such as Google Safe Browsing or VirusTotal.

The feature set is also deliberately narrow. It does not include certificate or TLS metadata, domain registration age, or homograph and typosquatting distance to well-known brand names, all of which are known signals in the phishing-detection literature and are absent here purely because they require a network lookup that the current lexical-only design was built to avoid. Extending the pipeline with an optional, cached WHOIS or certificate-transparency lookup would likely catch a class of attacks — freshly registered look-alike domains — that hostname entropy alone does not reliably distinguish from an unusual but legitimate brand name. Finally, the analyst dashboard uses a single hard-coded credential pair and an open cross-origin policy, both acceptable for a single-user local demonstration but requiring standard hardening — proper authentication, TLS, and a scoped CORS policy — before any multi-user or internet-facing deployment. Combining this URL-level system with a flow-level detector trained on a dataset such as CICIDS2017 [1], where a suitable network vantage point is available, is a promising direction for covering both layers at once.

VIII. CONCLUSION

This paper presented a hybrid rule-based and machine-learning system that intercepts navigation at the browser and blocks phishing, malware, ransomware, and DGA-style malicious destinations in real time, without requiring network-level packet capture. A Random Forest classifier trained on eight lexical and structural URL features reached 95.83% accuracy and a 95.52% F1-score on a held-out test set, at an inference cost of under 6 ms per request, well within the latency budget of a normal page navigation. More importantly, a real false-positive failure was found, measured, and fixed: restricting an entropy rule from the full URL to the hostname alone, combined with removing a spurious pattern from the training corpus, reduced false blocks on ordinary websites from 7 of 10 to 1 of 12 with no loss of true detections. A live per-attack-type accuracy view complements the usual offline benchmark by scoring every



trained algorithm against real traffic as it happens, and a per-class confusion matrix shows the two remaining error types clearly enough to prioritize future work: at the current dataset scale, missed detections (2 of 34) matter more than false alarms (1 of 38), which argues for growing the malicious-class training data before anything else. Taken together, these results suggest that a browser-level hybrid detector is a practical, lightweight complement to network-level intrusion detection for the individual users and small deployments that flow-based systems do not reach.

ACKNOWLEDGMENT

The authors thank Dr. Syeda Ayesha Unisa for her guidance and support throughout this project, and the Department of Computer Science and Engineering, RV Institute of Technology and Management, Bengaluru, for providing the resources and environment needed to carry it out.

REFERENCES

- [1] 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. Inf. Syst. Secur. Privacy (ICISSP), Funchal, Portugal, Jan. 2018, pp. 108–116.
- [2] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.
- [3] O. K. Sahingoz, E. Buber, O. Demir, and B. Diri, "Machine learning based phishing detection from URLs," *Expert Systems with Applications*, vol. 117, pp. 345–357, 2019.
- [4] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker, "Beyond blacklists: learning to detect malicious web sites from suspicious URLs," in Proc. 15th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD), 2009, pp. 1245–1254.
- [5] Y. Li, K. Xiong, T. Chin, and C. Hu, "A machine learning framework for domain generation algorithm-based malware detection," *IEEE Access*, vol. 7, pp. 32765–32782, 2019.
- [6] A. Bhatia, D. K. Vishvakarma, R. Kaushik, and A. Agrawal, "Entropy and likelihood-based detection of DGA generated domain names and their families," *Int. J. Security and Networks*, vol. 17, no. 3, pp. 129–141, 2022.
- [7] 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, 2nd Quarter 2016.
- [8] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, pp. 273–297, 1995.
- [9] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*. Boca Raton, FL, USA: CRC Press, 1984.
- [10] Chrome Developers, "Manifest V3," Google, [Online]. Available: <https://developer.chrome.com/docs/extensions/mv3/intro/>. Accessed: Sep. 2026.
- [11] D. Sahoo, C. Liu, and S. C. H. Hoi, "Malicious URL detection using machine learning: A survey," *arXiv:1701.07179*, 2019.
- [12] M. Aljabri et al., "Detecting malicious URLs using machine learning techniques: Review and research directions," *IEEE Access*, vol. 10, pp. 121395–121417, 2022, doi: 10.1109/ACCESS.2022.3222307.
- [13] M. Aljabri et al., "An assessment of lexical, network, and content-based features for detecting malicious URLs using machine learning and deep learning models," *Comput. Intell. Neurosci.*, vol. 2022, Art. no. 3241216, 2022, doi: 10.1155/2022/3241216.
- [14] S. Marchal, J. François, R. State, and T. Engel, "PhishStorm: Detecting phishing with streaming analytics," *IEEE Trans. Netw. Service Manag.*, vol. 11, no. 4, pp. 458–471, Dec. 2014.
- [15] S. Yadav, A. K. K. Reddy, A. L. N. Reddy, and S. Ranjan, "Detecting algorithmically generated malicious domain names," in Proc. 10th ACM SIGCOMM Conf. Internet Measurement (IMC), Melbourne, Australia, Nov. 2010.
- [16] A. Khraisat, I. Gondal, P. Vamplew, and J. Kamruzzaman, "Survey of intrusion detection systems: Techniques, datasets and challenges," *Cybersecurity*, vol. 2, Art. no. 20, 2019, doi: 10.1186/s42400-019-0038-7.
- [17] D. Gümüşbaş, T. Yıldırım, A. Genovese, and F. Scotti, "A comprehensive survey of databases and deep learning methods for cybersecurity and intrusion detection systems," *IEEE Syst. J.*, vol. 15, no. 2, pp. 1717–1731, Jun. 2021, doi: 10.1109/JSYST.2020.2992966.
- [18] R. Vinayakumar, M. Alazab, K. P. Soman, P. Poornachandran, A. Al-Nemrat, and S. Venkatraman, "Deep learning approach for intelligent intrusion detection system," *IEEE Access*, vol. 7, pp. 41525–41550, 2019.
- [19] F. Pedregosa et al., "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.
- [20] C. E. Shannon, "A mathematical theory of communication," *Bell Syst. Tech. J.*, vol. 27, no. 3, pp. 379–423, Jul. 1948.
- [21] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in Proc. IEEE Symp. Security Privacy, 2010, pp. 305–316.