WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal Homepage: www.ijrpr.com ISSN 2582-7421

Adaptive API Abuse Detection with Hybrid Anomaly–Classifier Risk Scoring and Graduated Mitigation

Deepak Marlecha[1], Zarana J Shah[2], Aditi Singh[3], Dr. G. Paavai Anand[4]

[1–3] Undergraduate Students; [4] Faculty Guide

Department of Computer Science and Engineering, SRM Institute of Science and Technology, Vadapalani, Chennai, India

[1] RA2411003040043 [2] RA2411003040044 [3] RA2411003040003

[1] dm9184@srmist.edu.in [2] zj0286@srmist.edu.in [3] as8790@srmist.edu.in

ABSTRACT:

Most public APIs are still protected by fixed rules: a request-per-minute ceiling, a cap on failed logins, a blocklist. These rules are easy to reason about and easy to defeat. I observed that a scraper moving below the ceiling usually goes unnoticed but a normal power user who momentarily goes above that ceiling is stopped. This paper explains a gateway-side detector. The gateway-side detector swaps the threshold, for a risk score. Every client session is described by eleven features that are calculated over a sliding sixty-second window. These behavioural features, which include request cadence, endpoint diversity, error rate, authentication-failure rate, payload size, burstiness and the fraction of requests that target paths are used by the gateway-side detector to compute the risk score. An Isolation Forest trained only on benign sessions produces an anomaly score; an XGBoost classifier trained on labelled abuse profiles produces an attack probability and a category; a per-client exponential-moving-average profile measures how far the current window departs from that client's own history. We combine these signals with a term that measures how sensitive the endpoint is. The result is a score that ranges from zero to one hundred. That score determines four actions: allow, monitor, token-bucket throttle or a five-minute block. On a held-out set of 2,000 sessions the hybrid system reaches 99.15% accuracy with no false positives on 1,300 benign sessions, compared with 180 false positives for the Isolation Forest alone. We report the comparison with rule-based and single-model baselines, show that the risk distribution separates the two classes cleanly, and discuss at some length why the headline numbers should be read cautiously: the holdout set comes from the same synthetic generator as the training data, and the rule-based baseline scores far higher on it than it would on real traffic.

Keywords: API security; abuse detection; anomaly detection; Isolation Forest; XGBoost; behavioural profiling; risk scoring; rate limiting; API gateway; adaptive mitigation.

Introduction:

Almost every interaction with a modern application is an API call. Logging in, loading a product page, submitting a payment: each is a request to a backend endpoint, and each endpoint is reachable by anyone who can construct an HTTP request. This makes APIs a convenient target for automation that is not quite an exploit but is clearly unwanted — bulk scraping of catalogue data, credential stuffing against a login route, enumeration of hidden administrative paths, or simply flooding a service until it degrades. OWASP tracks several of these under its API Security Top 10 [1], and most of them involve legitimate requests issued at an illegitimate rate or in an illegitimate pattern.

The standard defence is a rate limiter at the gateway. Every request, from each client is counted during each interval and any request that exceeds the pre-set limit is refused. Where the attack is a flood this works; where it is anything subtler it does not. The limit has to be set high enough that real users are not blocked, and anything set that high leaves room for an automated client to operate comfortably underneath. The limit also applies the same number to everyone, so a partner integration that legitimately issues thousands of requests and a browser session that issues a few dozen are judged by the same rule.

Background and Motivation:

What distinguishes abusive traffic from normal traffic is rarely a single quantity. I noticed that a scraper can stay under the rate limit. That same scraper may hit the same two endpoints hundreds of times with almost no variation. A credential-stuffing script may issue requests at a human pace but accumulate authentication failures that no human would. I notice that a fuzzer touches distinct paths and collects 404 responses. The fuzzer keeps collecting 404 responses from each path. These are patterns across several features over a window of time, which is exactly what a rule on one counter cannot see and what a learned model can.

At the same time, a purely learned detector has its own failure mode. A supervised classifier recognises the attack types it was shown and nothing else; an unsupervised anomaly detector flags anything including the busy but honest user. Neither alone gives the operator what they want, which is to let normal traffic through untouched, slow down traffic that looks wrong, and stop traffic that is clearly hostile, with the confidence to act proportionately rather than binary. That requirement pushed the design here: two models that work together a model, for each client and a scoring method that changes their results into a ranked answer.

Problem Statement:

Given a stream of HTTP requests arriving at an API gateway, each tagged with a client identifier, path, method, response status, latency and payload size, the task is to assign every client session a risk score in [0, 100] that is low for ordinary use and high for automated abuse, to do so with latency small enough to sit in the request path, and to translate the score into one of four actions whose severity increases with the score. A secondary requirement is that the decision be explainable: when a request is throttled or blocked, the system should be able to say which signals contributed and by how much.

Technical Challenges:

Labelled API abuse data is scarce. Public datasets we could find are small. Public datasets cover attack types. Public datasets do not share a feature schema. The Kaggle API access dataset [2] describes 34,423 sessions in terms of request cadence, sequence length and endpoint uniqueness but carries no error or authentication fields; the Cisco–Ariel ATRDF challenge data [3] contains 4,282 individual HTTP requests with payload-level attack labels such as SQL injection, which is a different problem from behavioural abuse. Neither covers scraping, credential stuffing or fuzzing as session-level behaviours. We therefore built a controlled generator for those profiles and used the public data to anchor what benign cadence looks like, a compromise discussed further in Section 5.

Feature sizes can vary by times. The time between requests is counted in seconds. Usually does not go beyond one minute. A flood can raise the number of requests to thousands in the period. Using a z-score scale would squeeze the extreme sessions and shrink all other data toward zero. Therefore a median-and-IQR scale is chosen instead. New customers have no data. The behaviour-deviation measure depends on a baseline, for each customer, which is missing for the few requests. A fallback is needed that does not let new customers slip through unchecked nor does it label every newcomer as a threat. Some fallback is required that neither waves new clients through nor treats every newcomer as suspicious.

Finally, the weights that combine the signals and the thresholds that separate the four actions are design choices. They are not learned, and the paper does not claim they are optimal; they were set by hand to produce sensible scores on the worked examples in Section 3.6 and then left alone.

Contributions:

The components are standard. What this project adds is a specific, working combination: an eleven-feature session summary computed in a sliding window at the gateway; a two-model core in which an unsupervised detector and a supervised classifier cover each other's blind spots; a per-client exponential-moving-average profile with a population prior for cold start; a transparent additive risk formula whose per-term contributions are returned with every decision; and a four-tier mitigation policy with a token-bucket throttle between "allow" and "block". We also include a comparison against three baselines on a common holdout set and an explicit account of where that comparison is and is not informative.

Related Work:

Rate Limiting and Rule-Based Protection:

Token-bucket and leaky-bucket limiters are decades old and remain the default in API gateways and web application firewalls. They are stateless beyond a counter, apply uniformly to all clients, and have no notion of what a client normally does. The OWASP API Security project lists two dangers: unrestricted resource consumption and broken authentication. It says that rate limits alone do not stop automated threats that stay under the limit. Unrestricted resource consumption is a risk that can run out of server capacity. Broken authentication is a risk that lets attackers take over accounts. Rate limits are part of the solution because some attackers can use automated tools that stay below the limit and still cause harm.

Anomaly Detection for Web Traffic:

Liu, Ting and Zhou [4] introduced the Isolation Forest, which isolates points by random axis-aligned splits and scores them by the average depth at which they become isolated; outliers need fewer splits. It requires no labels, trains in near-linear time and scores in microseconds, which suits a gateway. Behaviour-based anomaly detection for web applications has been around, for a time [5] usually focusing on the way requests are made or the way parameters are distributed. The weakness common to these approaches is that "unusual" and "malicious" are not the same set, and the false positives land on the most active legitimate users.

Supervised Attack Classification:

Gradient-boosted trees, and XGBoost in particular [6], are the usual choice for tabular security data because they handle mixed scales, skewed distributions and feature interactions without much preparation and expose feature importances for inspection. Their limitation in this setting is the obvious one: they classify into the categories they were trained on. Traffic that is abusive in a new way is assigned to whichever known class it most resembles, which may be "benign".

Behavioural Baselines and Risk Scoring:

Per-entity baselines — comparing a user to their own history rather than to the population — are common in fraud and insider-threat detection but less so in API gateways, where state per client is a cost. Risk-scoring approaches that combine detectors into one number appear in commercial bot-management products though the combination rules are rarely published. The formula in Section 3.6 is in that spirit, with the difference that its weights and terms are stated.

Research Focus:

The literature offers each piece: cheap unsupervised scoring, accurate supervised classification, per-user baselines, graded enforcement. The question this project addresses is whether a small, explainable combination of them, built from public and generated data and runnable inside a single Python gateway process, performs better than any one piece alone — and what evidence a student project can realistically provide for that.

Methodology:

System Overview:

Figure 1 shows the processing path for a singular request. A FastAPI middleware intercepts the request, records its metadata to an SQLite log, and appends it to a per-client window. The feature extractor recomputes the eleven session features for that window. The anomaly detector and attack classifier score the feature vector the profiler compares it to the client's baseline. The risk engine combines the results and selects an action. The whole path runs synchronously inside the request; in our measurements model inference takes under two milliseconds with the SQLite write being the step.

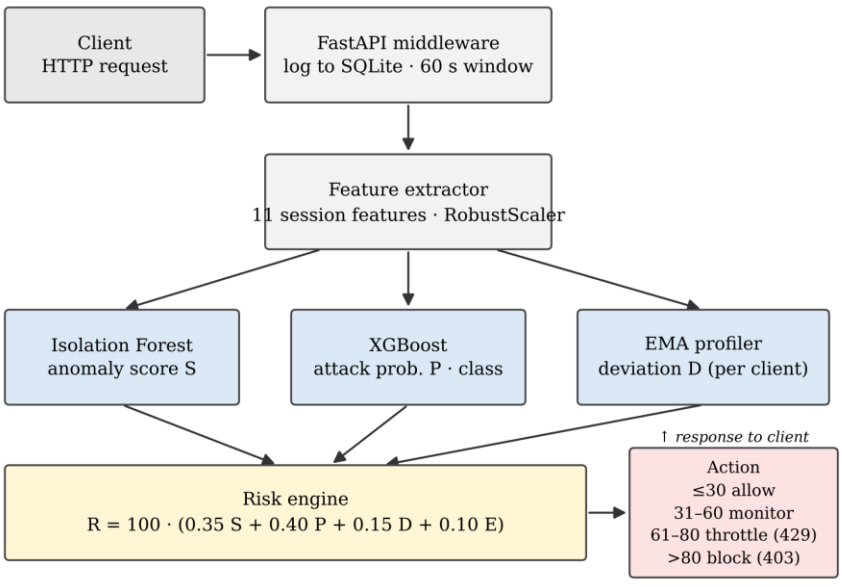


Fig 1. Request path through the gateway. Shaded boxes are the learned or stateful components; the risk engine is a fixed formula.

Data Sources:

Three sources were used, summarised in Table 1. The Kaggle API access dataset provided benign cadence statistics; the ATRDF requests were analyzed to find sessions with payload-level attacks; and a controlled generator created labelled sessions for five behavioural profiles. The generator takes each profiles features, from hand-set distributions. A scraping bot for example has inter-request gaps, long sequences, low endpoint uniqueness and a low error rate while a fuzzer has high uniqueness and a 404-dominated error rate. Profile proportions were 65% normal, 10% scraping, 10% credential stuffing, 8% fuzzing and 7% burst flooding.

Table 1. Data sources and their role in training

Source	Records	Labels	Used for
Kaggle API access behaviour [2]	34,423 sessions	anomaly flag	Benign cadence for Isolation Forest (8,000 sampled)
Cisco–Ariel ATRDF [3]	4,282 requests	attack type	Payload-attack profiles; reference only
Controlled generator (ours)	5,000 sessions	5 profiles	Isolation Forest (3,000 benign); XGBoost (all)

The Kaggle data lacks six of the eleven features. For the benign sessions drawn from it, those fields were filled from distributions chosen to look like quiet normal traffic (error rate uniform in [0, 0.05], at most one authentication failure, burst ratio in [1, 2], and so on). This is a known weakness: part of the "real" benign training data is half-synthetic. The Isolation Forest therefore learned normal behaviour from 11,000 sessions of which 8,000 have real cadence features and generated auxiliary features, and 3,000 are fully generated. The XGBoost classifier was trained on the 5,000 generated sessions only, because no public dataset labels the behavioural profiles we wanted.

Feature Extraction:

Each client is identified by IP address and session token. Requests in the last sixty seconds form the window, and the following eleven features are recomputed on every request: mean inter-request interval; number of requests in the window (sequence length); session age in minutes; ratio of distinct paths to total requests (uniqueness); count of distinct paths; fraction of 4xx/5xx responses; count of 401/403 responses; mean server latency; mean request payload size; fraction of requests to paths tagged sensitive (login, admin, payment); and burst ratio, defined as the peak request rate in any five-second slice divided by the window's mean rate. Features are scaled with a RobustScaler fitted on the benign training sessions, so a value of zero corresponds to the benign median and a unit step to one interquartile range.

Anomaly Detector and Attack Classifier:

The Isolation Forest uses 200 trees with a contamination parameter of 0.04. It has a raw decision function that is adjusted to an anomaly score S between 0 and 1. The values that are above 0.55 are not part of the main group of normal training sessions. The values that are, above 0.55 are not part of the main group of normal training sessions. The XGBoost classifier uses 250 trees of depth 5 with learning rate 0.08 and is trained as a five-class problem over the generator's profiles. At inference the probability mass on the four attack classes is summed to give an attack probability P, and the argmax class is kept as the category label for the explanation. Both models load from joblib files at gateway start.

Behavioural Profiling:

For each client the profiler maintains an exponential moving average and variance of five features (inter-request interval, sequence length, error rate, authentication failures, burst ratio) with smoothing factor $\alpha = 0.15$, updated after every request. The deviation of the current window is the mean absolute z-score against this baseline, squashed through a logistic function so that a mean z of about 2.5 maps to 0.5:

$$D = 1 / (1 + \exp(-0.9 \cdot (\bar{z} - 2.5))) \qquad (1)$$

A client with fewer than three recorded requests has no usable baseline, so the profiler substitutes population priors — fixed means and standard deviations typical of benign traffic — and a slightly more lenient logistic ($\bar{z}$ offset 2.0, slope 0.8). The effect is that a new client is judged against "normal users in general" until it has a history of its own.

Risk Engine and Mitigation:

The four signals are combined additively with fixed weights and scaled to [0, 100]:

$$R = 100 \cdot (0.35 \cdot S + 0.40 \cdot P + 0.15 \cdot D + 0.10 \cdot E) \qquad (2)$$

where E is the sensitive-endpoint ratio from the feature vector. Two compounding rules handle the cases where the linear form underweights. If the window holds five or more authentication failures and P is higher than 0.5 0.20 is added to the composite before scaling. Otherwise if P is higher than 0.85 and S is higher, than 0.60 0.15 is added. The composite is clipped to [0, 1]. The attack probability carries the largest weight because it is the most specific signal; the anomaly score is next because it is the only signal that can fire on an unseen pattern; behavioural deviation and endpoint sensitivity are modifiers rather than detectors.

The score selects an action. Below 30 the request is forwarded unchanged. From 31 to 60 it is forwarded with an audit header and a log entry at elevated severity. From 61 to 80 the client is passed through a token bucket of capacity ten refilling at one token per second, and receives HTTP 429 when the bucket is empty. Above 80 the request is refused with HTTP 403 and the client identifier is placed in a five-minute cooldown during which every request

is refused. Every response above 30 carries the four per-term contributions, the XGBoost category and the three features that most exceeded their benign medians, so the decision can be audited.

As a worked example, a window with S = 0.85, P = 0.95, D = 0.90, E = 1.0 and six authentication failures gives a composite of 0.2975 + 0.38 + 0.135 + 0.10 + 0.20 = 1.11, clipped to 1.0, so R = 100 and the client is blocked. A window with S = 0.10, P = 0.02, D = 0.05 and E = 0 gives R = 5.1 and is allowed.

Experimental Setup and Results:

Experimental Environment:

The system is written in Python 3 using FastAPI for the gateway, scikit-learn for the Isolation Forest and scaler, XGBoost for the classifier and SQLite for logging. Training and evaluation ran on a laptop CPU. All random seeds are set to numbers. The number 42 is used for the training data and, for the models. The number 999 is used for the holdout generator. The evaluation script writes a JSON file of metrics and four figures; every number below is taken from that file.

Testing Methodology:

A holdout set of 2,000 sessions (1,300 benign, 700 abusive) was generated with the controlled generator under a seed disjoint from training. Four systems were scored on it. The rule-based baseline flags a session if sequence length exceeds 100, or error rate exceeds 0.40, or authentication failures reach 5, or burst ratio exceeds 3.0 — a reasonable approximation of a tuned WAF rule set. The standalone Isolation Forest flags sessions with S > 0.55. The standalone XGBoost flags sessions with P > 0.50. The proposed system flags sessions with R > 60, i.e. those that would be throttled or blocked. For ROC curves the continuous score of each system was used.

We report accuracy, precision, recall, F1 and ROC-AUC with the abusive class as positive. Precision is the metric the operator cares most about here, since a false positive is a legitimate user throttled or blocked; recall tells us how much abuse gets through at the chosen operating point.

Benchmark Results:

Table 2. Performance on the 2,000-session holdout set

System	Accuracy	Precision	Recall	F1	ROC-AUC	FP	FN
Rule-based thresholds	99.35%	100.0%	98.14%	0.991	1.000	0	13
Isolation Forest only	86.65%	77.3%	87.57%	0.821	0.960	180	87
XGBoost only	99.95%	100.0%	99.86%	0.999	1.000	0	1
Proposed hybrid (R > 60)	99.15%	100.0%	97.57%	0.988	1.000	0	17

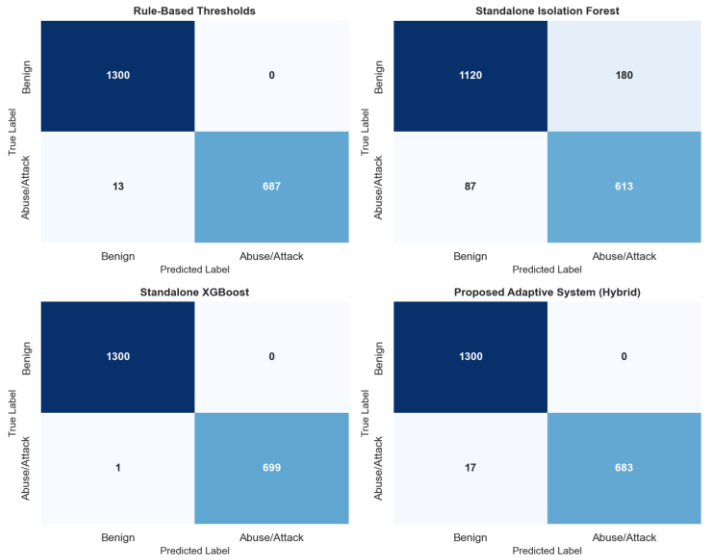


Fig 2. Confusion matrices for the four systems on the holdout set. Rows are true labels, columns are predicted labels .

Three things are standing out. The Isolation Forest alone is the only system with false positives, and it has many: 180 of 1,300 benign sessions, or 13.8%, which would be unusable at a gateway. The hybrid removes all of them, because the attack probability and behavioural deviation terms are both low for those sessions and pull the composite below the throttle threshold. Second, XGBoost alone is slightly better than the hybrid on recall, missing one abusive session against seventeen. This is expected: the hybrid's threshold of $R > 60$ is more conservative than $P > 0.5$, and the anomaly and deviation terms dilute a confident classifier verdict when the session does not also look anomalous. Third, the rule-based baseline is far stronger than its reputation, and that is the point we return to in Section 5.

ROC and Risk Distribution:

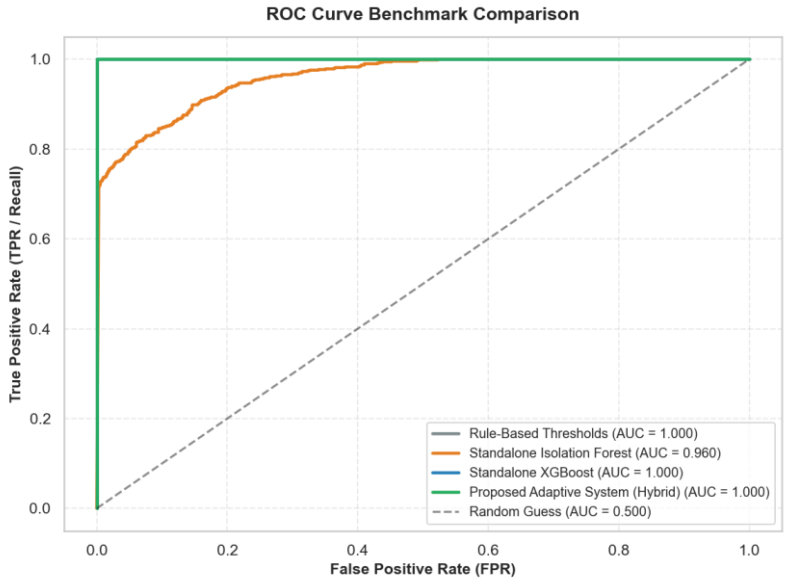


Fig 3. ROC curves. The rule-based, XGBoost and hybrid curves coincide at the top-left corner.

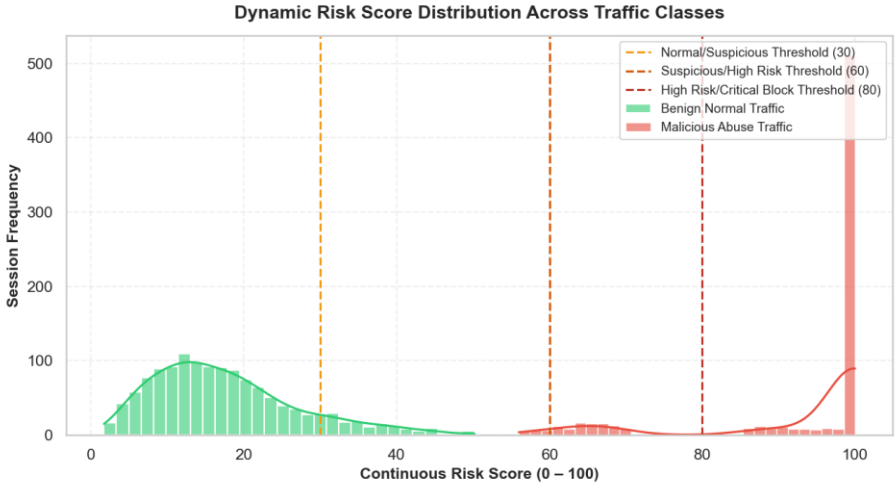


Fig 4. Distribution of the hybrid risk score for benign (green) and abusive (red) holdout sessions, with the three action thresholds marked.

The risk distribution in Figure 4 is the most useful view of the hybrid system's behaviour. Benign sessions mostly cluster around 12 forming a peak. The distribution has a tail that stretches out to, about 45. No session goes above 60. Abusive sessions pile up at 100, with a secondary group between 60 and 70 — these are sessions where one model is confident and the other is not, which is exactly the band the throttle tier was designed for. The 17 missed sessions sit in the 45–60 gap, where they are logged and monitored but not slowed. A threshold at 50 would recover most of them without touching the benign tail on this data; we left it at 60 because the benign tail on real traffic will certainly be longer than on generated traffic.

Feature Importance:

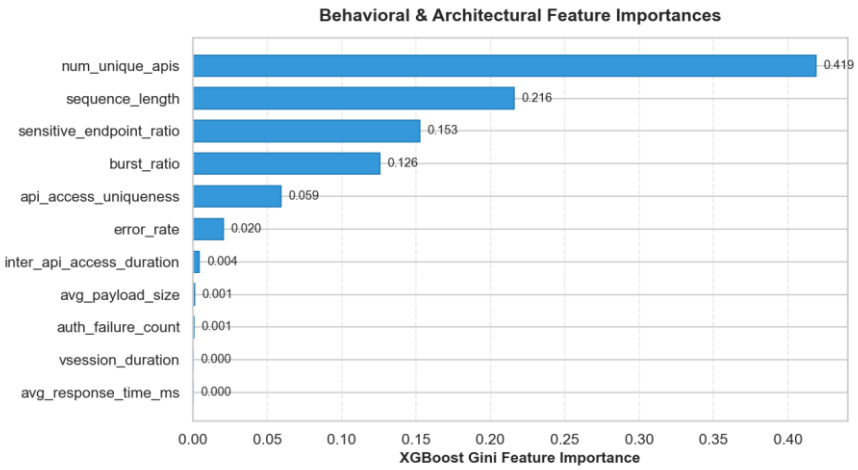


Fig 5. Gini importance of the eleven features in the XGBoost classifier.

The classifier relies mainly on the number of distinct endpoints (0.42), sequence length (0.22), sensitive-endpoint ratio (0.15) and burst ratio (0.13). Error rate and authentication-failure count, which one might expect to dominate, contribute almost nothing. The explanation is in the generator: credential stuffing is the only profile with high authentication failures, and it is also the only profile with a sensitive-endpoint ratio near one, so the classifier can identify it from the latter and never needs the former. On real traffic, where sensitive-endpoint ratio is noisier, we would expect the failure count to take over some of that role. This is a reminder that importances describe the training distribution as much as the problem.

Discussion:

Key Findings:

The system does what it was designed to do on the data it was designed on. Combining an anomaly detector with a classifier and a behavioural baseline eliminates the anomaly detector's false positives entirely while keeping recall above 97%, and the resulting score separates the classes with a wide margin. The graded response is also doing real work: the sessions in the 60–80 band in Figure 4 would be blocked outright by a binary system and allowed outright by a lenient one; throttling them is the proportionate choice.

The more interesting finding is the one that looks like a weakness. The rule-based baseline, which the project set out to replace, scores 99.35% accuracy and zero false positives. That is not evidence that rules are sufficient; it is evidence that the generated attack profiles are coarse. A scraping bot that issues 300 requests in a minute, or a fuzzer with a 60% error rate, trips a simple threshold trivially. The low-and-slow attacker that motivated the project — the one pacing at 55 requests per minute against a 60 limit — is not in the holdout set, so the experiment cannot show the hybrid catching what the rules miss. It only shows that the hybrid matches the rules on easy cases while also producing a calibrated score, an explanation and a graded action, which the rules do not.

Limitations and Validity:

The training and holdout sessions come from the same generator with different seeds. They share the same distributional assumptions, so a model that learns those assumptions will do well on both. This is the central threat to validity, and the ceiling-level numbers in Table 2 should be read as "the pipeline works end to end" rather than as an estimate of production performance. Real traffic will have a longer benign tail, more overlap between profiles, and abuse that does not match any of the four generated patterns.

The benign training data for the Isolation Forest is partly synthetic even where it derives from a real dataset, because six of the eleven features had to be filled in. The anomaly detector's sense of "normal" for those features is therefore ours, not the data's. The ATRDF data, though real, describes payload-level injection attacks rather than behavioural abuse and ended up contributing little beyond a sanity check.

The risk weights, compounding rules and action thresholds are hand-set. We did not run a sensitivity analysis and cannot say how much the results depend on them. The behavioural profiler was exercised only indirectly in the benchmark: the holdout sessions are independent, so the deviation term was approximated from the anomaly score rather than computed from a client history. Its contribution in a live deployment is untested.

Evaluation used a single train/holdout split and a single seed per stage. With generated data the variance is likely small, but we have not measured it.

Future Work:

The most important next step is evaluation on captured traffic from a real API with ground truth, or at minimum a generator that includes deliberately sub-threshold attackers so that the hybrid's advantage over rules can be demonstrated rather than argued. Replaying the holdout as a time-ordered stream through the live gateway would also exercise the profiler properly. On the modelling side, the hand-set weights could be replaced by a small logistic layer fitted on held-out sessions, and the four-class XGBoost could be extended as new abuse profiles are added. Sequence models over the raw request stream are a natural extension but would need to justify their latency at the gateway.

Conclusion:

We built a gateway-side API abuse detector that scores each client session on a 0–100 scale by combining an unsupervised anomaly detector, a supervised attack classifier, a per-client behavioural baseline and an endpoint sensitivity term, and that responds with one of four actions of increasing severity. On a 2,000-session holdout set the hybrid achieves 99.15% accuracy with no false positives, removing the 180 false alarms produced by the anomaly detector alone, and its risk distribution separates benign from abusive traffic with a clear margin. The same experiment shows a simple rule set performing nearly as well, which we take as a statement about the generated data rather than about rules; the hybrid's practical advantages are its calibrated score, its per-decision explanation and its graded response, and demonstrating its detection advantage over rules on realistic sub-threshold abuse remains the work ahead.

References:

[1] OWASP Foundation, “OWASP API Security Top 10,” 2023. https://owasp.org/www-project-api-security/

[2] Tangodelta, “API access behaviour anomaly dataset,” Kaggle, 2021. https://www.kaggle.com/datasets/tangodelta/api-access-behaviour-anomaly-dataset

[3] Ariel University Cyber Innovation Lab and Cisco, “API security challenge dataset (ATRDF),” GitHub, 2022. https://github.com/ArielCyber/Cisco_Ariel_Uni_API_security_challenge

[4] F. T. Liu, K. M. Ting, and Z.-H. Zhou, “Isolation forest,” in Proc. 8th IEEE International Conference on Data Mining (ICDM), pp. 413–422, 2008.

[5] C. Kruegel and G. Vigna, “Anomaly detection of web-based attacks,” in Proc. 10th ACM Conference on Computer and Communications Security (CCS), pp. 251–261, 2003.

[6] T. Chen and C. Guestrin, “XGBoost: A scalable tree boosting system,” in Proc. 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 785–794, 2016.

[7] F. Pedregosa et al., “Scikit-learn: Machine learning in Python,” Journal of Machine Learning Research, vol. 12, pp. 2825–2830, 2011.

[8] S. Ramírez, “FastAPI,” 2018–. https://fastapi.tiangolo.com/

[9] V. Chandola, A. Banerjee, and V. Kumar, “Anomaly detection: A survey,” ACM Computing Surveys, vol. 41, no. 3, pp. 1–58, 2009.