TwinGuard: An Adaptive Digital Twin for Real-Time HTTP(S) Intrusion Detection and Threat Intelligence

Anonymous Author(s)

Abstract

HTTP(S) based attacks against IoT and web infrastructure are increasingly evasive, adaptive, and diverse, making them difficult to detect using static rules or fixed signatures. In this paper, we present TwinGuard, a lightweight digital twin framework for adaptive intrusion detection and behavioral profiling at the application layer. TwinGuard comprises three layers: a physical layer that ingests real-world traffic from globally distributed honeypots, a virtual layer that mirrors session behavior using a trie based sequence model and dual machine learning classifiers, and an intelligence layer that provides attacker fingerprinting and hierarchical pattern-based labeling. TwinGuard monitors unknown sequence surges and classifier degradation in real time and adapts through a sliding window retraining mechanism. We processed over 4 million honeypot records collected during a 26-day evaluation. TwinGuard maintains 90% classification accuracy with over 98% pattern match rate during stable periods, and we detect more than 60,000 previously unseen attack sequences. TwinGuard demonstrates high adaptability by successfully integrating traffic from a heterogeneous honeypot, recovering from a 42% unknown rate and a 20–30% accuracy drop within one retraining cycle. Beyond detection, TwinGuard provides attacker fingerprinting and hierarchical labeling, uncovering diverse toolsets and behavioral shifts across user agents and cloud infrastructures. We argue that existing detection systems fall short in coping with the pace and variability of modern application-layer attacks. In contrast, TwinGuard offers a scalable and adaptive approach that not only detects novel threats in real time but also contextualizes them through behavioral fingerprints and hierarchical labeling.

CCS Concepts

• **Security and privacy** → **Intrusion detection systems**; **Web application security**; • **Networks** → *Network monitoring*.

Keywords

Intrusion Detection, Digital Twin, Honeypot, Security

ACM Reference Format:

Anonymous Author(s). 2025. TwinGuard: An Adaptive Digital Twin for Real-Time HTTP(S) Intrusion Detection and Threat Intelligence. In *Proceedings of Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security (CCS '25).* ACM, New York, NY, USA, 20 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Modern HTTP(S) infrastructures face a growing wave of adversarial behavior ranging from automated scanners to evasive payload delivery [10, 29]. These threats are increasingly hosted in cloud environments [25], leveraging flexible infrastructure and polymorphic tooling to bypass conventional defenses. Traditional intrusion detection systems (IDS) struggle to keep pace with these changes, especially under application-layer attacks that deviate from known patterns or evolve dynamically in real time [22]. To address this, we propose a digital twin framework where the physical layer, consisting of real world attacker behavior captured by honeypots, is mirrored by a virtual model that evolves continuously through sequence tracking and adaptive detection.

While prior work often depends on static detection, offline adaptation [20, 26] or concept drift detection [2, 23], our approach introduces a unified and lightweight framework for adaptive intrusion detection. It captures evolving behaviors, supports real-time updates, and enables long-term behavioral insights.

In this shifting threat landscape, defenders face two compounding challenges: (1) the lack of high quality labeled data for supervised intrusion detection, and (2) the operational need for lightweight and real-time monitoring. These constraints are especially critical in IoT and cloud edge environments, where traffic diversity, limited resources, and the emergence of novel attack variants require systems that are both adaptive and interpretable. A minimal overhead detection approach that captures behavioral shifts without relying entirely on ground truth is essential for addressing these gaps.

These two motivating factors give rise to four concrete challenges in designing a practical HTTP(S) intrusion detection system. Firstly, the diversity of attacker tools and user-agent behaviors complicates traffic interpretation, especially in the absence of consistent labeling. Secondly, application layer attacks often use subtle or obfuscated payloads, evading traditional signature-based detection and further complicating reliable categorization. Thirdly, maintaining detection accuracy over time requires continuous adaptation to emerging threats with minimal overhead, especially under real-time constraints. Finally, as adversaries increasingly exploit cloud infrastructure, distinguishing malicious behavior from benign automation calls for contextual understanding and behavioral profiling beyond static rule matching. These challenges correspond directly to the motivating constraints: the first two arise from limited ground truth, while the latter two reflect the demands of lightweight, adaptive monitoring in dynamic environments.

In this paper, we present TwinGuard, an adaptive intrusion detection system designed for HTTP(S) traffic monitoring at scale. TwinGuard is built on a digital twin framework, where a virtual representation of attacker behavior evolves alongside real-world observations. To ensure consistent labels, TwinGuard uses its probabilistic trie to cross verify model predictions when ground truth is available, and to identify contextual mismatches that may indicate labeling errors or inconsistencies. In unlabeled settings, it highlights novel patterns that require further annotation, serving both verification and discovery functions. To address obfuscated

CCS '25, Taipei, Taiwan
2025. ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

requests and behavioral drift, the trie model tracks unfamiliar or evolving request patterns over time, enabling early detection of attack shifts and offering timely feedback for retraining.

TwinGuard combines a dual model architecture focused on adaptability and efficiency: a core machine learning classifier handles general intrusion detection, while a sliding window strategy incrementally updates the model without large-scale centralized retraining. This ensures lightweight operation with minimal overhead which is ideal for real-time deployments. For long-term behavior analysis, TwinGuard aggregates summarized features from past sessions to identify gradual changes in attacker strategies without the need to retain full traffic logs.

This design addresses the earlier challenges as follows: (1) To mitigate the lack of labeled data, the trie enables consistency checks and highlights novel patterns for annotation. (2) To detect obfuscated requests and behavioral drift, TwinGuard leverages sequence-level pattern tracking alongside machine learning classifier detection. (3) To maintain accuracy with minimal overhead, the system updates incrementally via sliding windows, avoiding full-scale retraining. (4) For behavioral profiling, historical summaries capture attacker evolution over time without the need for full session logs. In general, this paper makes the following contributions:

- We present TwinGuard, a lightweight adaptive intrusion detection system for HTTP(S) traffic that combines a trie tree structure with machine learning classifiers to support efficient real-time detection and periodic retraining.
- We introduce a digital twin framework that tracks evolving adversarial behavior through unknown sequence detection, sliding window retraining, and long-term attacker profiling.
- We implement a semantic fingerprinting pipeline to quantify behavioral divergence across user agent groups and cloud hosted adversaries, enabling explainable group level analysis.
- We evaluate TwinGuard on 3.3 million labeled HTTP(S) sessions from a global honeypot network, and further validate it with an independently deployed honeypot platform with 0.8 million sessions, demonstrating adaptability and generalization across heterogeneous traffic sources.
  We make our software available at https://anonymous.4open.science/r/TwinGuard-B41A

2 Assumptions, Goals, and Challenges

Threat Model. *Victim.* We assume the victim to be an application-layer service—typically a web-based interface exposed over HTTP(S), which may run on cloud-hosted servers, resource-constrained IoT devices or edge gateways. These services are targets for a range of threats including probing, scanning, credential abuse, configuration exploits, and payload delivery attempts.

Adversary. The adversaries are remote attackers issuing HTTP(S) requests with potentially malicious intent. In our dataset, they originate from over 200 distinct IPs across major cloud providers. These actors emulate various tools, user-agents, and scanning techniques to probe, exploit, or fingerprint the monitored systems. We also assume that adversaries can dynamically alter tactics over time—changing headers, URIs, or payloads to evade detection.

Threats. TwinGuard is designed to detect application-layer attacks that manifest within HTTP(S) communication. These include but are not limited to reconnaissance scans, exploit attempts and command injection via malicious URIs or bodies. Our detection operates at the level of individual HTTP sessions and aggregates insights over time. TwinGuard is not responsible for mitigation or enforcement decisions (e.g., blocking), but provides real-time detection and contextual intelligence.

Goals. While honeypots provide rich ground truth, leveraging them for real-time, interpretable, and adaptive intrusion detection remains challenging. Digital twin modeling offers a bridge between capture and detection, yet a unified approach is still lacking. To guide our system design, we frame the following key research questions:

RQ1. How can we efficiently detect evolving and previously unseen application-layer attacks in HTTP(S) traffic from large-scale honeypot networks? This goal emphasizes the importance of generalization across diverse and novel attack patterns. Our trie-based real-time model is designed to catch known sequences and escalate novel ones with minimal latency.

RQ2. To what extent can lightweight, self-adaptive components maintain intrusion detection performance amid evolving attack patterns? TwinGuard incorporates an adaptive engine that periodically retrains using a sliding window strategy. We aim to understand how well TwinGuard preserves detection accuracy in the presence of changing attacker behavior.

RQ3. How can an intrusion detection system offer contextualized threat intelligence beyond simple alerts? Beyond binary intrusion labels, TwinGuard aims to offer rich, multi dimensional threat insights through hierarchical taxonomy mapping and behavioral fingerprinting across attacker groups (e.g., cloud providers, user agents).

Non-Goals. TwinGuard is currently designed for HTTP(S) application layer traffic. Extending support to other protocols such as SSH, Telnet, or FTP is beyond the scope of this work. Similarly, while TwinGuard is compatible with future plugins, including real IoT testbed traffic or production web server logs—this paper focuses solely on data derived from honeypot-based environments. Although TwinGuard performs real-time monitoring logic, our evaluation operates on time-bounded data retrieved from a global cybersecurity focused nonprofit organization, which does not support continuous streaming. TwinGuard can be deployed in self-hosted or cloud environments to enable true real-time adaptation to incoming web traffic.

3 TwinGuard Design

3.1 TwinGuard Architecture

TwinGuard adopts a layered architecture that mirrors the structure of a digital twin system, progressing from raw data capture to adaptive monitoring to high-level behavioral reasoning. As shown in Figure 1, the architecture is divided into three conceptual layers:
Physical Layer. This layer represents the sensory interface of the system. It ingests real-time HTTP(S) traffic collected from a globally distributed honeypot network that emulates common web facing services. Focusing solely on attacker-facing traffic, this layer isolates adversarial behavior in a controlled environment and serves as TwinGuard 's entry point.

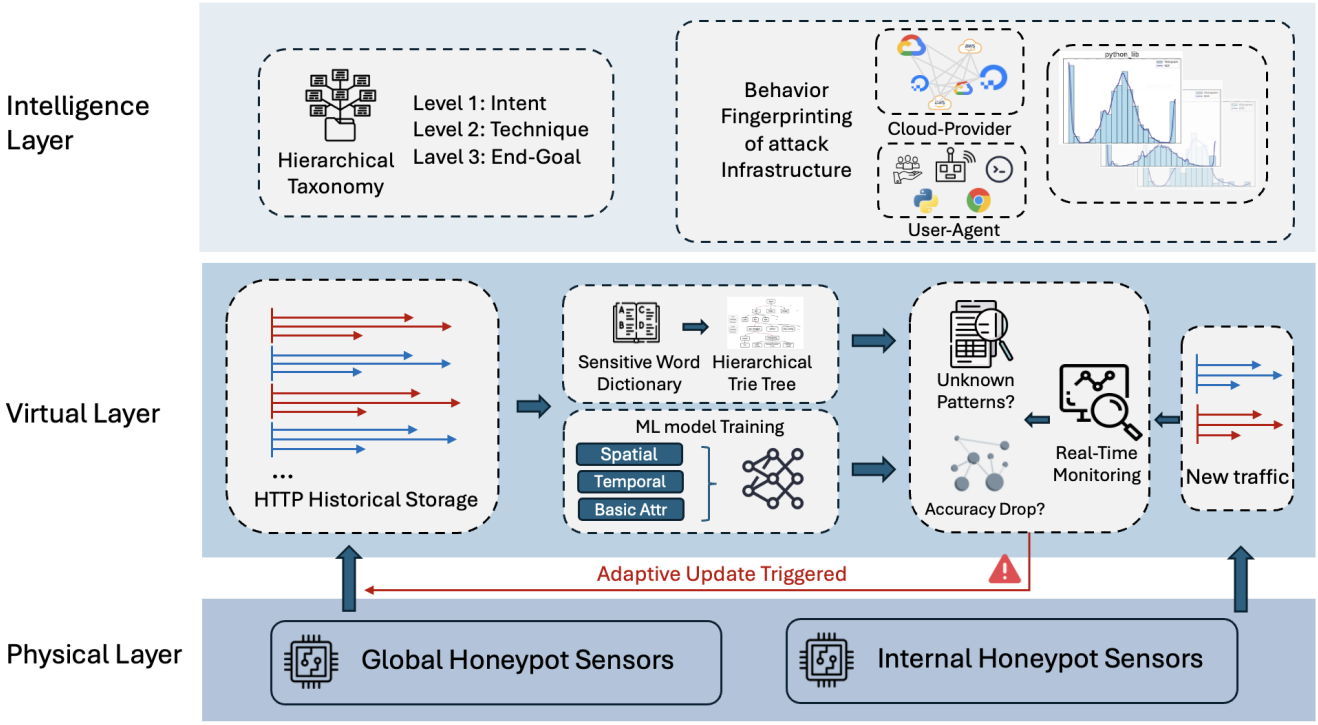


Figure 1: TwinGuard architecture for the proposed adaptive digital twin framework. The design is organized across three conceptual layers: (1) *Physical Layer* captures real-world HTTP(S) traffic via honeypot sensors; (2) *Virtual Layer* performs real-time monitoring and adaptive intrusion detection using trie-based path modeling, machine learning classifiers; (3) *Intelligence Layer* supports threat interpretation via hierarchical taxonomy mapping and attacker fingerprinting. The digital twin is instantiated in the virtual layer as an evolving behavioral replica of incoming attack traffic, supporting model adaptation, unknown pattern detection, and behavior profiling.

Virtual Layer. The virtual layer forms the core of the digital twin by transforming raw HTTP(S) traffic into structured session representations suitable for modeling and detection. It parses and tokenizes each request to extract normalized feature vectors that reflect both semantic signals and statistical traits.

These structured representations are continuously fed into two parallel behavioral models. A trie-based model captures sequential request paths and performs real-time pattern matching to identify deviations from previously observed behaviors. In parallel, a machine learning classifier utilizes spatial, temporal, and categorical features to assign intrusion category labels to each session. Supporting both models, an adaptive engine monitors classification drift and the emergence of novel request patterns, triggering updates when performance degrades or unfamiliar behaviors are detected.

Intelligence Layer. At the top of the architecture, the intelligence layer provides contextual interpretation and profiling of detected intrusion behaviors. It incorporates a hierarchical taxonomy to classify attacks into structured categories and subcategories, capturing both technical exploit techniques and their behavioral goals. This taxonomy is applied to labeled sessions to support finer-grained attribution and threat understanding. In parallel, TwinGuard performs attacker fingerprinting by grouping behavioral patterns based on user agent identifiers and cloud infrastructure sources.

Overall, these abstractions translate raw detection results into meaningful insights, facilitating adaptive defense strategies and longitudinal tracking of evolving adversary behaviors.

3.2 Data Lifecycle: from Honeypot to Detection and Profiling

The TwinGuard system follows a multi-layered data lifecycle, organized around three architectural layers. This layered progression supports a comprehensive view of intrusion behavior from initial capture to adaptive monitoring and long term behavioral profiling.

Data Gathering. Operating at the physical layer, a globally distributed honeypot network captures live HTTP(S) traffic from adversarial sources. These honeypots emulate vulnerable application endpoints and log essential session metadata. This layer functions as the sensory interface to the external threat landscape, providing raw behavioral signals for downstream analysis.

Adaptive Detection. The virtual layer transforms captured traffic into structured features, forming a digital twin of attacker interactions. Two real-time detection modules operate in parallel: a trie-based model for path matching and a machine learning classifier trained on session-level features. An adaptive retraining module

monitors detection performance and triggers model updates when accuracy degrades or novel patterns emerge. A sliding window approach ensures agility, discarding outdated data to keep models aligned with evolving attack behaviors.

Long-Term Profiling. Beyond real-time detection, TwinGuard aggregates session-level outputs over time to construct persistent attacker profiles based on user-agent traits and infrastructure sources. This enables alignment with a hierarchical intrusion taxonomy and reveals how behaviors evolve across sources and time. Combined with short-term adaptation, this profiling supports sustained situational awareness.

Together, these layers form an end-to-end pipeline for adaptive intrusion detection, combining reliable data collection, responsive detection, and interpretable profiling to support resilient and scalable threat monitoring.

4 Physical Layer: Honeypot Networks and Data Acquisition

The physical layer of TwinGuard serves as the entry point for real world HTTP(S) traffic and acts as the sensory interface for capturing adversarial behavior. We leverage two complementary honeypot networks that emulate web facing services to monitor malicious activity. One network is externally maintained and provides structured HTTP session data, while the other is reserved for internal experimentation and evaluation of heterogeneous input.

4.1 Primary Honeypot Network

Our primary data source is a globally distributed honeypot infrastructure maintained by a nonprofit cybersecurity organization focused on IoT threat monitoring and defense. It operates over 190 sensors deployed across more than 25 countries, hosted across four major cloud providers. Using a proprietary emulation framework, the infrastructure mimics diverse IoT devices and web-facing services, enabling the capture of real-world adversarial behavior across globally distributed networks.

The infrastructure continuously captures structured HTTP(S) interaction logs, including request metadata, authentication attempts, command execution traces, and malware retrieval activity. Each session is enriched with contextual information such as IP geolocation, autonomous system attribution, and third-party malware scan results, facilitating detailed threat characterization. Sessions are categorized into one of three classes: intrusion, attempt, or scan. While benign scanning may occasionally occur, the system is designed to prioritize visibility into suspicious or hostile activity.

Our deployment integrates daily logs from this infrastructure into the adaptive detection pipeline to enable continuous evaluation of evolving traffic behavior. Although the broader platform supports multi-protocol monitoring, we focus exclusively on HTTP(S) due to its prevalence in web-based threats and the availability of high-fidelity session features. The consistency, granularity, and global scale of this dataset make it foundational to our ability to support generalizable, real-world intrusion detection.

4.2 Internal Honeypot Platform

For the internal platform, we adopt an adaptive HTTP honeypot designed to emulate web interfaces of IoT devices. The honeypot platform consists of two main components: the attack observation component and the HTTP response collection component.

The attack observation component emulates a variety of IoT devices by accepting communications on all TCP ports and returning different responses based on the port number and HTTP path of the request. Specifically, the attack observation component identifies an HTTP response corresponding to a given port number and request path from a set of pre-stored HTTP response data and returns it to the client as a response. At the same time, it records incoming HTTP requests from attackers as observation data. This component is also responsible for malware collection. Specifically, when an incoming HTTP request contains commands such as curl or wget, it extracts the URLs targeted by those commands and attempts to download malware from the specified URLs.

The HTTP response collection component is triggered when the attack observation component observes access to a previously unseen combination of port number and HTTP path. In such cases, it attempts to collect the corresponding HTML file from the Internet. To do so, it searches the surrounding IP addresses of the source host that issued the request, looking for devices that respond to the same port and HTTP path, and collects their responses. This mechanism enables the honeypot nodes to expand the variety of device behaviors they can emulate.

5 Virtual Layer: Real-Time Monitoring and Adaptive Detection

To enable scalable, real-time intrusion detection, TwinGuard abstracts application-layer traffic using a trie-based sequence monitor, a machine learning classifier, and an adaptive retraining engine. This layer structures raw HTTP(S) sessions, detects behavioral deviations, and evolves with changing traffic patterns.

5.1 Trie-Based Path Sequence Monitoring

TwinGuard employs a trie based model for real time monitoring of HTTP request sequences. This approach offers several advantages. Firstly, it is memory efficient and scalable, as the trie shares common prefixes across sequences, reducing redundancy and enabling fast lookups. Secondly, it provides human interpretable structure, where each node represents a semantic step such as method, status, or keyword, allowing intuitive tracing of attacker paths. Thirdly, it supports unsupervised insight by flagging unfamiliar or low frequency patterns in unlabeled settings, and verifying predictions or revealing misclassifications in labeled environments. Finally, it enables low latency assessment by performing immediate semantic classification and anomaly flagging, serving as an efficient and transparent pre filtering mechanism before engaging heavier classifiers.

To construct the trie tree, each HTTP session is first transformed into a normalized, structured representation that enables consistent pattern matching and probabilistic reasoning. Each session is abstracted into a structured tuple (method, status, uri keywords), allowing TwinGuard to reason about behaviors at a more semantic level while reducing sparsity.

Sensitive Word Extraction. A dynamic sensitive keyword dictionary is generated from a frequency analysis of URI and body tokens. Tokens are extracted via regex-based tokenization and counted using a configurable frequency threshold (default = 20). Only frequently occurring tokens are retained, improving generalization and reducing noise.

Structured Path Representation. Each HTTP request is transformed into a tuple representation such as:

$$(method \rightarrow status \rightarrow URI\ keywords)$$

This format is used to represent attack paths for both training and detection. For instance, a request like:

GET /app/login_panel?user=admin&theme=dark HTTP/1.1

is parsed into tokens such as [app, login, panel, user, admin, theme, dark]. If *login* and *admin* are recognized sensitive keywords, this request is abstracted into the structured path:

$$(GET \rightarrow 200 \rightarrow login,admin)$$

This abstraction helps reduce URI noise and improve generalization across similar attack patterns while preserving important semantic cues.

Probabilistic Trie Construction. TwinGuard builds a hierarchical, probabilistic Trie, where each node corresponds to a sequence component (method, status, URI keyword group). Each node maintains the following metadata:

- Count: frequency of visits to the node in historical data
- Probabilities: normalized distribution over attack categories (e.g., scan, attempt, intrusion-control)
- Next Step: frequency of transitions to child nodes in the sequence

This structure enables fast lookup for real-time classification and efficient anomaly detection by flagging previously unseen paths.

5.2 Intrusion Detection Classifier

TwinGuard integrates a lightweight machine learning classifier to assign coarse-grained intrusion labels (i.e., *scan*, *attempt*, and *intrusion-control*) to each HTTP(S) session in real time. The classifier is trained on a diverse feature set designed to capture the structural, semantic, and contextual properties of each request.

Complementarity with Trie Monitoring. The machine learning classifier serves as a general-purpose intrusion detection component trained on labeled session features. In contrast, the trie model provides an interpretable view of structured request paths by aggregating common behavior patterns. It offers direct estimates of likely intrusion categories by comparing new paths to previously observed sequences. While operating independently of labels, the trie supports unsupervised detection and reveals emerging traffic patterns. The utility of the trie in unlabeled settings is further validated through its strong empirical correlation with the supervised classifier, as demonstrated in Figure 4 in Section 7.2.2.

Feature Engineering. We extract major groups of features for each HTTP(S) session to support real-time classification:

- **Basic HTTP Features:** Method and status codes, user-agent profiling across six categories (e.g., browsers, bots), and header-related flags.
- **Content Embeddings:** 50-dimensional GloVe-based semantic embeddings are generated from tokenized URI and body content.
- **Encoding and MIME Indicators:** Binary indicators for supported content-encoding formats and MIME types; rarity and presence flags are included for robustness.
- **Spatial Features:** Source host's ASN, organization, and country information for contextual attribution.
- **Temporal Features:** Session timing patterns including hour of day, weekday, and logarithmic session duration.

These features capture the semantic, structural, and contextual dimensions of each request. The full feature pipeline is visualized in Appendix C Figure 9.

Classifier Implementation. We use both Random Forest (RF) and XGBoost (XGB) models to assess robustness across modeling approaches. While RF tends to favor structured categorical features such as user-agent type, connection behavior, and encoding preferences, XGB places greater emphasis on the semantic content captured by embedded URI and body tokens. This contrast allows TwinGuard to maintain a balance between protocol-driven logic and deep semantic representations, offering resilience against both shallow and obfuscated intrusion attempts.

Real-Time Classification. Once trained, the classifier is encapsulated within the pipeline and deployed to operate in real time. Each incoming session is parsed, embedded, and classified with minimal latency, forming the core detection path in the virtual layer.

5.3 Adaptive Sliding-Window Update Engine

The adaptive update module in TwinGuard continuously monitors performance degradation and structural novelty within the HTTP(S) traffic stream. It does so by integrating both a probabilistic trie-based matcher and a supervised classifier into a unified pipeline that triggers retraining only when necessary. The workflow logic is detailed in Appendix G Algorithm 2.

Pipeline Overview. Figure 2 illustrates the adaptive retraining process. It begins with constructing the Trie Tree and training/testing splits using a historical window. The testing portion is used to evaluate the model and compute a baseline accuracy, which is later used to detect accuracy drops in subsequent traffic. After evaluation, if any of the trigger thresholds are met, all components are retrained using the new window.

Trigger Conditions. TwinGuard supports both supervised and unsupervised retraining mechanisms depending on the availability of ground truth labels. During controlled evaluation, we leverage labeled historical data from the primary honeypot source to compute classification accuracy. Both a significant drop in accuracy and a surge in unknown patterns beyond a configured threshold signal potential concept drift and prompts model retraining.

In live settings where labels are unavailable, the system monitors structural novelty using the trie model. If the proportion of request paths that fail to match any known sequences exceeds a specified threshold, this indicates emerging behavior and triggers retraining. This dual mechanism ensures adaptive responsiveness under both labeled and unlabeled conditions.

Generally the system monitors two signals:

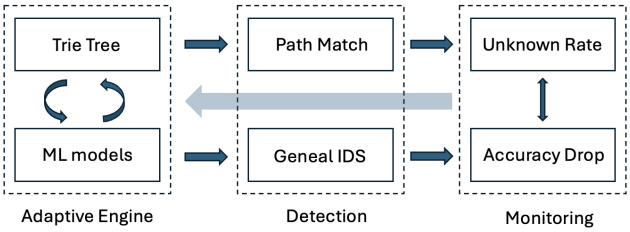


Figure 2: Overview of the adaptive intrusion detection workflow. TwinGuard monitors model performance and triggers updates when classification accuracy drops significantly or when the rate of unknown Trie path matches exceeds a predefined threshold.

- *Accuracy Drop:* The difference between the current model's accuracy on recent test data and its baseline accuracy (measured at training time).
- *Unknown Pattern Rate:* The proportion of HTTP request sequences that do not match any known path in the trie structure, calculated as a percentage.

Update Actions. When either trigger condition is met, the system retrains the following components using the new window:

- The trie tree, reconstructed using updated session data and refreshed keyword generalization.
- The sensitive word dictionary, reflecting token frequency across URI and body fields.
- The machine learning classifier, incorporating new features.

This design ensures the digital twin remains responsive to shifts in attacker behavior while minimizing unnecessary retraining.

6 Intelligence Layer: Intrusion Labeling and Attacker Attribution

6.1 Hierarchical Pattern-Based Intrusion Labeling

To contextualize detected intrusions and uncover attacker intent, we implement a hierarchical taxonomy that maps raw HTTP request patterns to structured semantic labels. The taxonomy is organized across three levels:

- **Level 1 (Intrusion Category):** groups intrusions by their stage in the attack lifecycle, such as exploit attempts, post exploitation and delivery.
- **Level 2 (Technique):** captures the specific method used, such as SQL injection, command and control callback, or shell upload.
- **Level 3 (Behavioral Goal):** defines the attacker's intended outcome, such as database access, remote control or persistent access.

Labels are assigned via regex matching over HTTP fields, linking request attributes to known adversarial behaviors. This taxonomy-driven method supports real-time enrichment and aligns with threat modeling practices like MITRE ATT&CK [14], where tactics and techniques are organized by intent and context.

Compared to prior systems such as YODA [9], C-FRAME [3], and Aristaeus [11], which use static analysis or flat labeling based on bot traffic, TwinGuard applies a hierarchical taxonomy built from protocol level request features. This approach enables more interpretable and extensible intrusion classification. The complete taxonomy is listed in Table 1, with pattern definitions provided in Appendix E Table 5.

6.2 Attacker Behavioral Fingerprinting

To understand the behavioral diversity of adversaries, we fingerprint attack patterns by grouping HTTP sessions according to cloud origins and user agent category. Each group is profiled using aggregated HTTP features, density estimation, and taxonomy alignment, revealing how distinct tooling or infrastructure impacts observed intrusion behaviors.

6.2.1 Feature Representation and Grouping. Each HTTP session is encoded as a normalized feature vector $\mathcal{X} = \{x_1, x_2, ..., x_n\}$, where all fields are embedded or converted into numerical values to support consistent profiling and comparison. The feature set includes:

- *URI embeddings*, capturing the semantic structure of request paths and facilitating similarity analysis across different targets.
- *HTTP headers*, such as user agent, accept, and connection fields, representing both standard and custom client behaviors.
- *Connection metadata*, including request method, response status code, body length, and frequency based indicators.

Sessions are grouped based on:

- *User agent categories*, including browser clients, command line tools, python libraries, automated scanners, and custom built clients. Representative detailed classification for each group are provided in Table 3 in Appendix B.
- *Cloud service providers*, where attackers are hosted on four major cloud platforms, these providers are referred to as Cloud A to Cloud D throughout our analysis.

This structured representation allows for the generation of fingerprint profiles that reflect behavioral consistency or variation across groups.

6.2.2 Signature Profiling and Visualization. To analyze behavioral consistency within each group and intrusion category, we construct a fingerprint vector by averaging the normalized session features defined previously. For a given group g and category c, let $D_{(g,c)}$ be the set of feature vectors $\mathcal{X} = \{x_1, x_2, ..., x_n\}$ associated with that combination. The group category fingerprint is defined as:

$$F_{(g,c)} = \frac{1}{|D_{(g,c)}|} \sum_{x \in D_{(g,c)}} x$$

We construct fingerprints for combinations of user agent groups and cloud service providers, which serve as attacker source indicators. (1) cloud providers, where $g \in$ {Cloud A, Cloud B, Cloud C, Cloud D}, and (2) user agent categories, where $g \in$ {browser, cli tool, python lib, scanner bot, custom client, missing, other}. These groupings support the creation of compact behavioral profiles across different attacker origins and tooling strategies.

Table 1: Hierarchical Pattern-Based Intrusion Taxonomy

Intrusion Category	Technique	End Goal
Exploit Attempts	File Inclusion (LFI/RFI)	Code Execution
	Misconfiguration Exploit	Priv. Esc. / Info Leak
	REST/JSON Abuse	Data Leak / Enumeration
	SQL Injection (SQLi)	DB Access / Bypass
	Command Injection	Code Execution
	Denial of Service (DoS)	Resource Exhaustion
Web Shell Upload	Simple Shell Upload	Persistent Access
	Obfuscated Shell Upload	Stealth Backdoor
	Two-Stage Payload	Loader & Dropper
Post-Exploitation Activity	Botnet C2 Callback	Remote Control
	Cronjob Deployment	Persistence
	Spam Mailer Setup	Email Abuse
	Proxy/Relay Deployment	Lateral Movement
Delivery / Downloader	Direct Script Drop	Code Execution
	Drive-by Download / JS	User Exploitation
Obfuscated / Anomalous Behavior	Junk Payload Flood	Resource Exhaustion
	Unknown Pattern	Undiscovered Variant

To capture distributional characteristics, we apply kernel density estimation (KDE) to the aggregated feature values for each group and category:

$$\hat{f}_{(g,c)}(v) = \frac{1}{mh} \sum_{i=1}^{m} K\left(\frac{v - v_i}{h}\right)$$

where $K(\cdot)$ is the Gaussian kernel and h is the bandwidth parameter.

To quantify behavioral differences between two group category pairs (g_1, c) and (g_2, c), we compute the Jensen Shannon divergence between their KDE distributions:

$$\mathrm{JS}(P \parallel Q) = \frac{1}{2} \sum_{i=1}^{N} P_i \log_2\left(\frac{P_i}{M_i}\right) + \frac{1}{2} \sum_{i=1}^{N} Q_i \log_2\left(\frac{Q_i}{M_i}\right)$$

where $M_i = \frac{1}{2}(P_i + Q_i)$ and P_i, Q_i are the estimated densities at each point v_i.

The resulting profiles enable both visual and quantitative comparison, highlighting distinctions across attacker tooling and origins. These fingerprint vectors and their associated divergences reveal behavior consistency within groups and expose heterogeneity across user agent categories and cloud-based sources.

6.2.3 Integrate Taxonomy Mapping. To further examine the behavioral distinctions revealed in fingerprinting, we map the hierarchical intrusion taxonomy back to the associated user agent groups and cloud service organizations. Each intrusion label is split into its parent category and subcategory, allowing aggregation of intrusion distributions across different attacker sources.

This taxonomy alignment provides a structured verification of the behavioral patterns captured through feature based profiling. It supports the observations made in fingerprinting by highlighting group level tendencies across both user tooling and hosting infrastructure. By comparing the composition of the types of intrusion within each group, we reinforce the distinctions identified through fingerprint vectors and divergence analysis, which are detailed in Section 7.3.

7 Evaluation

7.1 Dataset and Labeling

7.1.1 Primary Dataset. Our primary dataset comprises 3,377,335 HTTP(S) session records collected over a 26-day period, from 2025-03-15 to 2025-04-09. The data is sourced from a globally distributed honeypot infrastructure maintained by a nonprofit cybersecurity organization. This infrastructure spans over 190 sensors deployed across more than 25 countries and is hosted across four anonymized cloud providers (Cloud A –Cloud D).

Each session is labeled as either *scan*, *attempt*, or *intrusion control*, and includes structured metadata fields detailed in Appendix A. To support behavioral fingerprinting, user-agent strings are grouped into functional categories, as described in Appendix B.

7.1.2 Internal Evaluation Dataset. To test generalizability under heterogeneous input, we incorporate data from an internal honeypot platform emulating IoT device interfaces. The dataset spans 2025-03-26 to 2025-03-31, with 19 nodes deployed across academic and cloud testbeds, generating 847,869 HTTP requests. Roughly

70% of fields align with our primary schema, allowing partial compatibility and realistic robustness testing.

7.1.3 Labeling Criteria. All HTTP(S) sessions are labeled as *scan*, *attempt*, or *intrusion-control* based on rule-based matching of request patterns, payload content, and endpoint semantics. The full logic is described in Appendix F. These labels serve as ground truth for training and evaluation throughout the detection pipeline.

7.2 Adaptive Monitoring

To evaluate the effectiveness of TwinGuard 's adaptive intrusion detection capabilities, we examine how the system responds to evolving HTTP(S) traffic patterns using sliding window configurations ranging from $w = 3$ to $w = 6$ days, with retraining triggered when the classification accuracy of both classifiers drops by more than 6.0% or the unknown pattern rate exceeds 3.0%. The evaluation focuses on four aspects: (1) classification accuracy, (2) correlation between anomaly rate and model degradation, and (3) adaptation to novel threats introduced by a new honeypot integration. Each evaluation highlights the performance of the adaptive engine, which combines a probabilistic trie tree, a sensitive word dictionary, and dual classifiers (RF and XGB) to support timely updates and robust detection in the presence of evolving adversarial behavior.

7.2.1 Accuracy and Unknown Rate Dynamics. Figure 3 illustrates accuracy and unknown sequence rate trends under window sizes $w = 3$ to $w = 6$, using both RF and XGB classifiers. Smaller windows (e.g., $w = 3$) yield quicker updates but show greater volatility, with XGB accuracy dropping to 36% on 03-24 despite a low unknown rate (<0.3%), likely due to noisy training data from 03-21. In contrast, larger windows ($w = 6$) maintain accuracy above 88% and exhibit more stable trends.

On 03-26, narrow windows ($w = 3$, $w = 4$) report over 22% accuracy loss and over 15% unknown rate, while $w = 6$ limits both to under 5%. A second widespread anomaly occurs on 04-06 across all windows, with accuracy drops up to 20% and unknown rates over 14%. Following retraining, the system recovers performance within a single update cycle, demonstrating strong adaptability to emerging behavioral patterns.

Overall, these results demonstrate the tradeoff between responsiveness and stability. TwinGuard sustains high accuracy while detecting over 60K unknown patterns through efficient adaptive retraining. Smaller windows discover more novel sequences, whereas larger windows trade early detection for improved robustness. Full statistics are provided in Appendix I, Figure 12.

7.2.2 Accuracy vs. Anomaly Correlation. To quantify the impact of anomalous traffic on model performance, we examine the relationship between the unknown sequence rate and the classification accuracy drop. As illustrated in Figure 4, a clear negative trend emerges, where higher proportions of unknown patterns are associated with larger declines in accuracy. This relationship is consistent across all window sizes and both RF and XGB models.

The pearson correlation coefficient is computed as $r = -0.4431$, with a p-value of 1.96×10^{-16}, indicating a statistically significant inverse relationship. These results empirically validate the effectiveness of our anomaly-triggered update mechanism.

7.2.3 Adaptation to Novel Threats via Internal Honeypot Integration. We evaluate TwinGuard 's adaptability by integrating the above internal dataset into the detection pipeline using a $w = 6$ sliding window(Figure 5). Upon ingestion, the unknown pattern rate surged to 42.11%, and accuracy dropped to 65.79% (RF) and 73.15% (XGB). After one retraining cycle, both models stabilized above 87% accuracy, demonstrating strong adaptability to previously unseen traffic behaviors. Detailed results are available in Appendix J, Table 7.

Summary. The adaptive evaluation shows that smaller windows offer higher responsiveness to behavioral shifts but introduce volatility and frequent retraining. Larger windows provide more stable performance by smoothing out transient anomalies, with $w = 6$ offering a favorable tradeoff. Correlation analysis confirms that surges in unknown patterns coincide with accuracy drops, validating the update mechanism. Integration of a heterogeneous honeypot source demonstrates strong generalization and recovery capabilities, reinforcing the robustness of TwinGuard.

7.3 Behavioral Fingerprinting Insights

We analyze attacker behavior by aggregating HTTP session features across user-agent categories and cloud infrastructure groups. For each group and intrusion category, we compute fingerprint vectors that summarize structural and semantic request patterns. Feature distributions are visualized using histograms and kernel density estimates (KDE), revealing group-specific traits. To assess consistency, we compute pairwise Jensen-Shannon divergence scores and map intrusion taxonomy labels to each group. This evaluation demonstrates TwinGuard 's capability to uncover attacker intent and tooling through behavioral signal analysis.

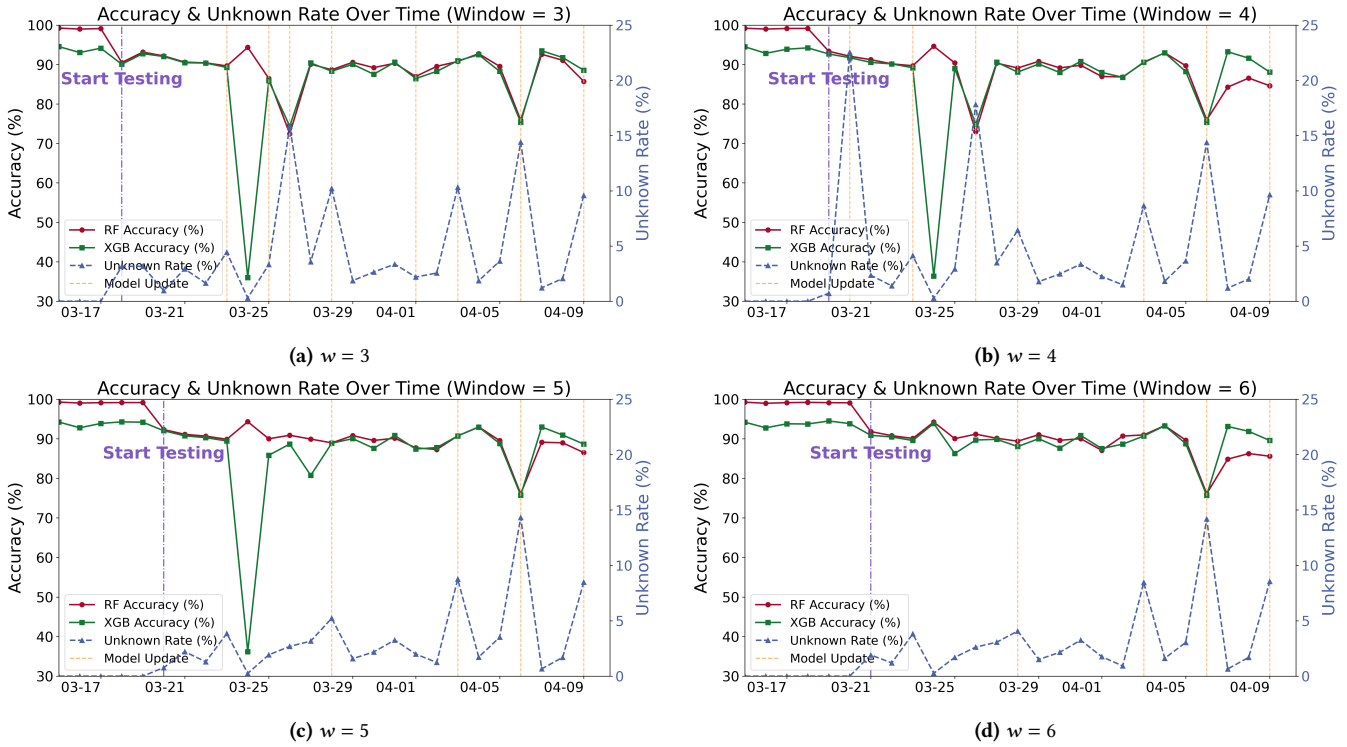
7.3.1 Feature Based Fingerprinting. We analyze per group behavioral profiles by aggregating HTTP session features across user agent categories and cloud service organizations. For each group and intrusion category, we compute fingerprint vectors by averaging all normalized session features, which include HTTP methods, status codes, headers, metadata, and URI embeddings.

To visualize these distributions, we plot histograms with overlaid KDE. Each subplot in Figure 6 corresponds to a group–category pair, showing the distribution of all normalized feature values. Appendix D Table 4 lists the full set of features. The *x-axis* represents different HTTP session features, and the *y-axis* indicates their normalized values across sessions.

These visualizations highlight distinct behavioral profiles. In Figure 6a, *python lib*, *scanner bot*, and *cli tool* show sharp, multipeak structures and high-frequency noise, suggesting varied automation tools. In contrast, *browser* and *custom client* display smoother, unimodal shapes, indicative of more stable or template-based interactions. *Missing* and *other* groups exhibit erratic, diffuse patterns, likely due to spoofed or ambiguous agent strings.

Cloud organization profiles in Figure 6b show consistent, symmetric shapes across providers. This suggests that attacks launched from cloud infrastructure may rely on standardized or shared tooling, reflecting centralized deployment practices.

7.3.2 Divergence Analysis. To quantify the behavioral consistency and variation observed in fingerprinting, we compute pairwise JS



(a) $w = 3$ **(b)** $w = 4$ **(c)** $w = 5$ **(d)** $w = 6$

Figure 3: Evaluation of adaptive retraining under sliding window sizes ($w = 3$ to $w = 6$) from 2025-03-15 to 2025-04-09. Classification accuracy (RF in red, XGB in green) and unknown sequence rate (blue) are tracked over time, with update triggers marked by vertical dashed lines (orange). Updates are initiated when a surge in unknown patterns coincides with a drop in accuracy. Smaller windows offer higher responsiveness to anomalies but exhibit greater volatility, while larger windows enable more stable and consistent adaptation.

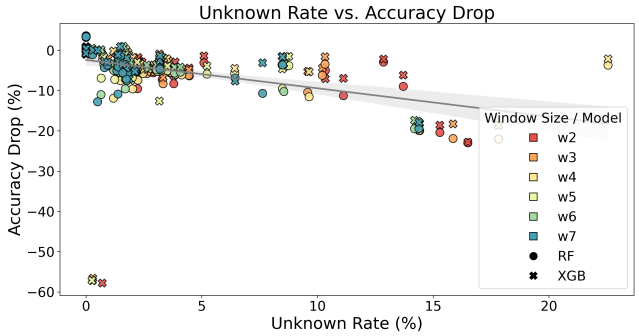


Figure 4: Correlation between unknown sequence rate and classification accuracy drop across all window sizes and models. Pearson correlation coefficient: $r = -0.4431$, $p = 1.96 \times 10^{-16}$.

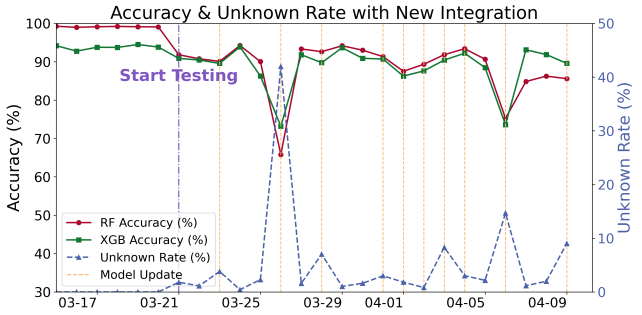
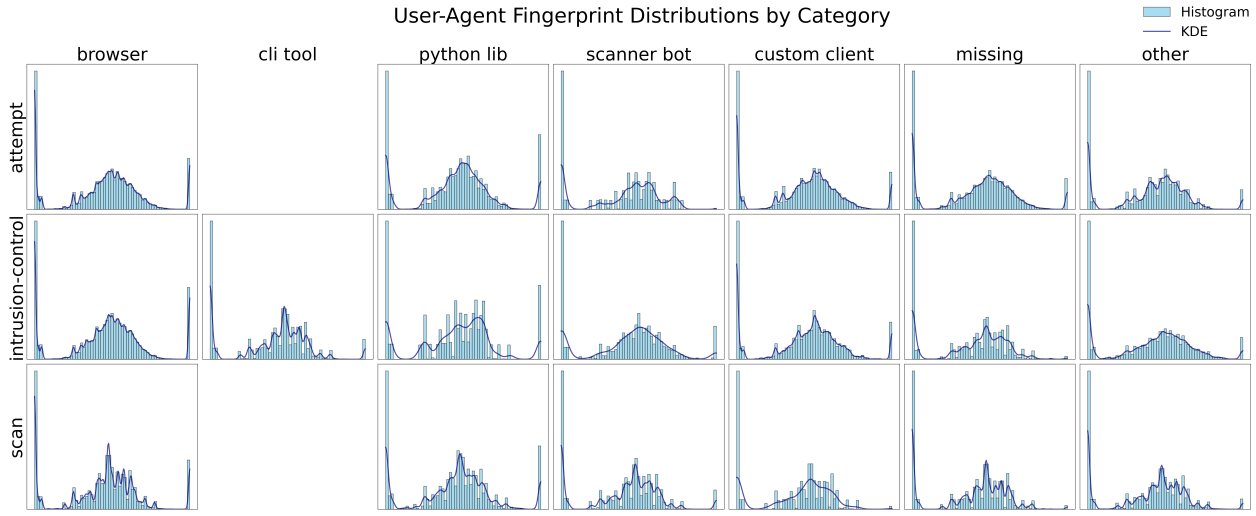


Figure 5: Adaptation to a new honeypot source under window size $w = 6$. A surge in unknown sequences and an accuracy drop is observed upon integration, followed by recovery after retraining.

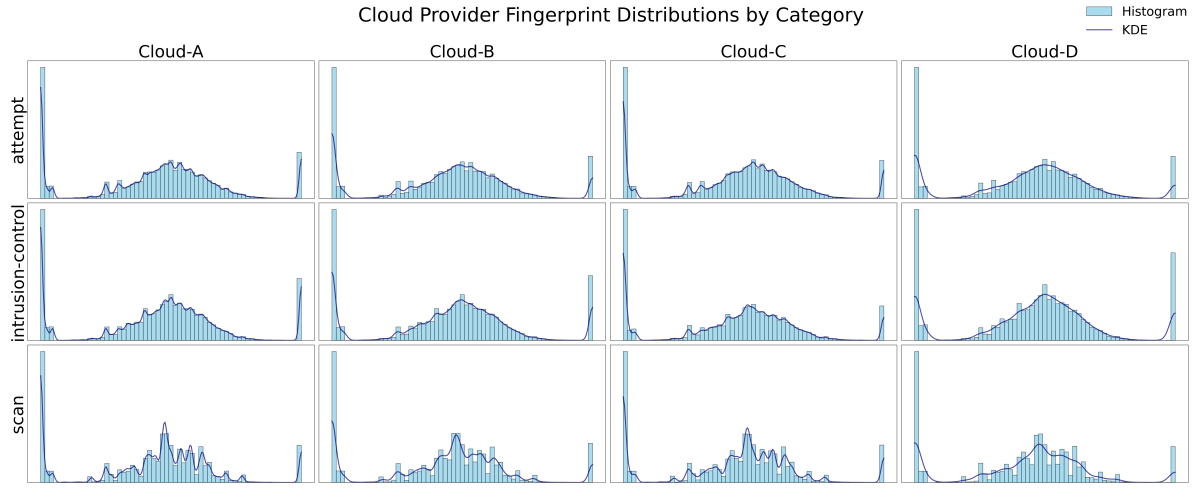
divergence between user agent and cloud provider categories under each intrusion type.

Separate thresholds are adopted to account for differing baseline variability across groups. For user agents, thresholds of 0.01 and 0.03 mark small to large divergence. For cloud organizations, lower thresholds of 0.005 and 0.015 are used due to more uniform behavior. These values are not universal, but reflect meaningful contrasts within the empirical range observed in our dataset.

User Agent Group Divergence. Figure 7a shows that scan behavior is largely consistent across user agent groups, with divergence



(a) User-agent group fingerprint distributions. Each cell shows the distribution of normalized HTTP features per attack category, revealing distinct behavioral patterns.



(b) Cloud organization fingerprint distributions. Distributions appear consistent across providers, suggesting shared infrastructure or standardized toolchains in cloud-hosted attacks.

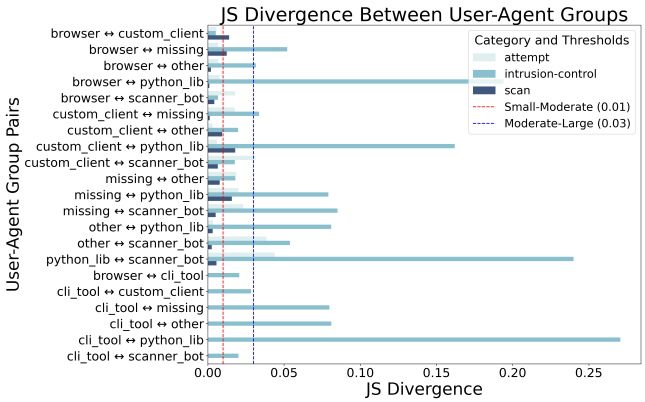
Figure 6: Fingerprint histograms with KDE overlays across attacker groups and categories. Each subplot visualizes the distribution of all normalized session features for a given group–category pair.

values typically below 0.01. In contrast, intrusion-control sessions exhibit higher divergence, especially among scanner bots, python libraries, and mixed user agents, indicating indicating the presence of diverse attack methods and possibly nonstandard clients.

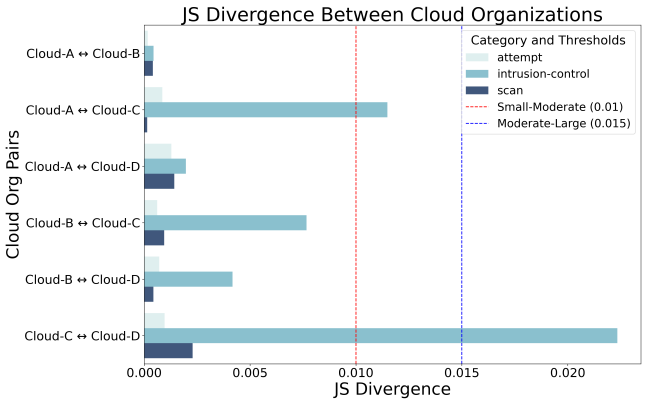
Cloud Organization Divergence. As shown in Figure 7b, cloud-level divergence remains low across categories. Minor variation appears in intrusion-control traffic involving Cloud C, but most values fall below 0.015, suggesting widespread use of shared toolchains.

7.3.3 Taxonomy Alignment. To further validate the behavioral distinctions captured through fingerprinting and divergence analysis, we analyze the distribution of intrusion subcategories across user agent groups and cloud service providers. This taxonomy-driven view enables a clearer interpretation of attack behaviors, identifying which techniques dominate within each group. Visualizations of parent category alignment and subcategory heatmaps are provided in Appendix H.

User Agent Mapping. Figure 8a shows the proportional distribution of attack subcategories across user agent groups. Browser and CLI tool sessions are concentrated in broad categories like exploit attempts and web shell uploads, reflecting traditional probing behavior. In contrast, python libraries and scanner bots demonstrate greater technique diversity, especially in misconfiguration exploits and file inclusion (LFI/RFI). The *missing* and *other* categories display highly irregular distributions, suggesting spoofed or unstable

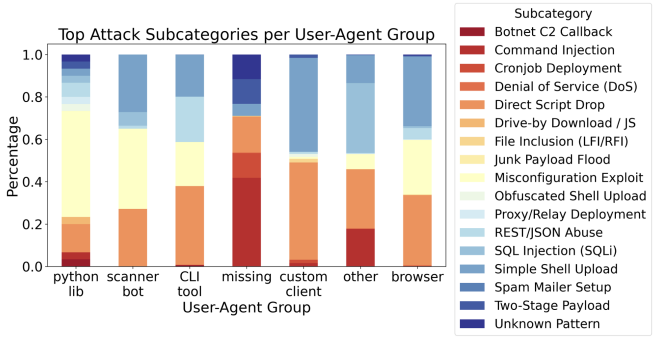


(a) JS divergence across user agent groups. Behavioral variation is more pronounced in intrusion-related sessions, especially among nonstandard user agents.

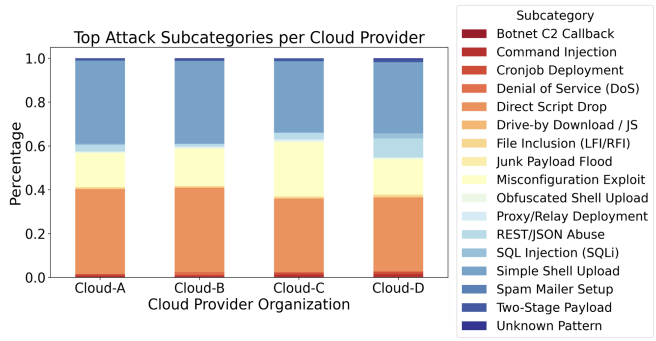


(b) JS divergence across cloud organizations. Most divergence values are low, suggesting shared tooling and minimal provider-specific variation.

Figure 7: Pairwise Jensen-Shannon divergence values across user agent groups and cloud organizations. While user agent diversity reveals significant behavioral variation, cloud-based sources tend to exhibit consistent patterns.



(a) User agent group wise distribution of intrusion subcategories.



(b) Cloud provider wise distribution of intrusion subcategories.

Figure 8: Stacked percentage distributions of intrusion subcategories aligned with user agent groups and cloud providers. The taxonomy breakdown verifies group-level distinctions captured in previous fingerprinting and divergence analysis.

automation strategies. These observations align with earlier divergence patterns from Section 7.3, reinforcing that user agent tooling significantly impacts the diversity of observed intrusion patterns.

Cloud Providers Mapping. Figure 8b As shown in Figure 8b, cloud originated traffic exhibits a largely uniform subcategory distribution, dominated by direct script drop, simple shell uploads, and misconfiguration exploits. However, subtle provider specific distinctions emerge: Cloud C shows elevated misconfiguration exploits and cronjob deployments, while Cloud D includes higher rates of proxy/relay setups and obfuscated shell uploads. These nuances suggest minor variation in deployment strategies, but overall consistency supports the notion that cloud-based attacks often rely on shared, automated toolchains.

Summary. The fingerprinting evaluation reveals that attacker behavior varies more significantly across user agent groups, while cloud providers exhibit more uniform patterns driven by standardized toolchains. User agents show higher diversity in feature distributions and intrusion subcategories, reflecting varied tooling and intent. Divergence analysis confirms these contrasts, and taxonomy alignment further validates the semantic differences. These findings underscore the effectiveness of profiling based on application layer behavior and user agent characteristics, rather than infrastructure origin.

8 Related Work

Digital twins in cybersecurity. The application of digital twins to cyber security has been explored in various contexts. A broad overview of their potential to simulate and analyze system behavior in cyber security settings has been presented [6]. The integration of digital twins into security-by-design practices has been proposed for industrial control systems (ICS) to enhance the resilience of cyber-physical infrastructure [13]. A digital twin-based ICS security framework has been developed, incorporating a machine learning-based intrusion detection system (IDS) and evaluated

through simulations of multiple attack scenarios [27]. Enhanced intrusion detection has also been achieved by integrating digital twins with an AutoML pipeline, including data preprocessing, feature engineering, model training, and drift detection through periodic monitoring [17]. The combination of digital twins with honeypot technology has been realized in a framework called DiTwinHon, designed to generate threat intelligence and analyze attacker behavior [16]. Additionally, a method for identifying network attacks based on spatio-temporal feature fusion within a digital twin environment has been proposed [28]. Despite these advancements, the application of digital twins remains largely confined to ICS environments, with most efforts focused on modeling physical systems and addressing network-layer threats. Little attention has been paid to application-layer security, especially in the context of HTTP-based web systems. Furthermore, no known implementation leverages real-time updates from honeypot data within a digital twin framework for dynamic threat modeling and response.

Intrusion Detection in Application Layer. A number of intrusion detection systems (IDS) targeting application-layer protocols have been proposed [4, 7]. Machine learning techniques have been employed to enhance anomaly detection capabilities in web applications, utilizing classifiers such as random forest, gradient boosting machines, and XGBoost [26]. Anomaly-based IDS approaches have also been developed and evaluated using data collected from real-world production web environments [15]. In the domain of HTTP URI anomaly detection, a comprehensive evaluation was conducted using multiple datasets based on 1-gram models [5]. The key distinction of our approach lies in the use of a digital twin to enable model adaptation, unknown pattern detection, and behavior profiling.

Attack Taxonomy and Labeling Methods. Various studies have focused on categorizing intrusion behaviors and developing labeling methods. Honeypot-based analyses have identified multi-stage attack workflows and grouped post-exploitation behaviors into categories such as spam mailers, IRC bots, and web shells [1]. Bot activities targeting CMS platforms like WordPress and Drupal have been observed through honeysite monitoring [11]. Semantic taxonomies have been used to classify plugin-based malware into distinct behavioral types, leveraging code structure and execution patterns [9]. CAPTCHA abuse has been systematically categorized using a behavior-driven labeling approach, identifying over 30 attack types across global traffic [3]. Web shells have also been analyzed at scale, revealing common functionalities and widespread use of obfuscation [24]. As demonstrated in Section 6, we employ a hierarchical taxonomy that maps raw HTTP request patterns to structured semantic labels.

Attacker Fingerprinting and Behavioral Profiling. Attacker profiling has been explored through the analysis of system event logs to capture behavioral patterns during process execution [19]. Graph-based models have also been applied; for example, the HINTI framework leverages graph convolutional networks to model various indicators of compromise (IoCs) for purposes such as vulnerability trend analysis and attack intent inference [30]. To counter identity obfuscation techniques like Tor and VPNs, browser fingerprinting has been proposed as a means of tracking attackers [8, 12]. Beyond HTTP, attacker behavior has also been studied using other protocols. For instance, Telnet negotiation data has been utilized for visitor profiling via open-source intelligence (OSINT) [21], while SSH-based profiling has examined actions such as configuration checks, password changes, and file downloads to infer attacker intent [18]. Our analysis focuses on JS divergence, user-agent group divergence, and cloud provider distribution. When integrated with the methods proposed in previous studies, our approach has the potential to enable more fine-grained attacker profiling.

9 Discussion

9.1 Implications

Policy and Ecosystem Implications. This work supports broader threat monitoring by providing a deployable, interpretable system grounded in the digital twin paradigm. TwinGuard can assist national web standards organizations, and threat intelligence platforms in surfacing unknown patterns and tracking behavioral shifts. These insights strengthen shared awareness of evolving attacker strategies across cloud and web ecosystems.

Operational Implications The modular and lightweight design of TwinGuard supports deployment in IoT gateways and edges. Its adaptive loop enables localized detection and fast response without relying on centralized infrastructure, making it well suited for real-time and resource-constrained environments.

9.2 Limitations

Protocol Scope. This work is scoped exclusively to HTTP(S) application layer traffic. While the underlying honeypot infrastructure supports other protocols, extending detection to these is left for future work.

Honeypot-Only Evaluation. All results are based on data collected from honeypot environments. Testing on an alternative honeypot platform demonstrates generalization potential, but further evaluation on real deployments remains future work.

Streaming Constraints. Our evaluation is based on time-bounded snapshots of attack traffic, due to the absence of continuous streaming from the data provider. However, real-time streaming can be supported in deployments that integrate directly with live network infrastructure.

10 Conclusion

TwinGuard presents a robust and adaptive framework for HTTP(S) based intrusion detection, achieving high accuracy while maintaining interpretability and responsiveness. Through a combination of trie based sequence monitoring and dual machine learning classifiers, the system sustains over 90% accuracy during stable periods and identifies more than 60,000 previously unseen attack sequences. A strong negative correlation between the rate of unknown patterns and classification accuracy supports the system's update mechanism, enabling timely retraining. The integration of traffic from a heterogeneous honeypot source further demonstrates TwinGuard 's adaptability, where a 42% spike in unknown patterns and a 30% accuracy drop are effectively mitigated through a single update cycle. Behavioral analysis reveals diversity across user-agent groups, while traffic originating from different cloud providers exhibits consistent patterns, suggesting the use of shared or templated toolchains. Together, these results highlight TwinGuard 's capacity for scalable, real-time monitoring and behavioral threat intelligence in evolving web environments.

References

[1] Davide Canali and Davide Balzarotti. 2013. Behind the scenes of online attacks: an analysis of exploitation behaviors on the web. In *20th Annual Network & Distributed System Security Symposium (NDSS 2013)*. n–a.

[2] Renjie Chu, Peiyuan Jin, Hanli Qiao, and Quanxi Feng. 2024. Intrusion detection in the IoT data streams using concept drift localization. *AIMS mathematics* 9, 1 (2024), 1535–1561.

[3] Hoang Dai Nguyen, Karthika Subramani, Bhupendra Acharya, Roberto Perdisci, and Phani Vadrevu. 2024. C-Frame: Characterizing and measuring in-the-wild CAPTCHA attacks. In *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 277–295.

[4] Jesús E. Díaz-Verdejo, Rafael Estepa Alonso, Antonio Estepa Alonso, and German Madinabeitia. 2023. A critical review of the techniques used for anomaly detection of HTTP-based attacks: taxonomy, limitations and open challenges. *Computers & Security* 124 (2023), 102997. doi:10.1016/j.cose.2022.102997

[5] Rafael Estepa, Jesús E. Díaz-Verdejo, Antonio Estepa, and German Madinabeitia. 2020. How Much Training Data is Enough? A Case Study for HTTP Anomaly-Based Intrusion Detection. *IEEE Access* 8 (2020), 44410–44425. doi:10.1109/ACCESS.2020.2977591

[6] Rajiv Faleiro, Lei Pan, Shiva Raj Pokhrel, and Robin Doss. 2022. Digital Twin for Cybersecurity: Towards Enhancing Cyber Resilience. In *Broadband Communications, Networks, and Systems*, Wei Xiang, Fengling Han, and Tran Khoa Phan (Eds.). Springer International Publishing, Cham, 57–76.

[7] P. García-Teodoro, J. Díaz-Verdejo, G. Maciá-Fernández, and E. Vázquez. 2009. Anomaly-based network intrusion detection: Techniques, systems and challenges. *Computers & Security* 28, 1 (Feb. 2009), 18–28. doi:10.1016/j.cose.2008.08.003

[8] Zhaopeng Jia, Xiang Cui, Qixu Liu, Xiaoxi Wang, and Chaoge Liu. 2018. Micro-Honeypot: Using Browser Fingerprinting to Track Attackers. In *2018 IEEE Third International Conference on Data Science in Cyberspace (DSC)*. 197–204. doi:10.1109/DSC.2018.00036

[9] Ranjita Pai Kasturi, Jonathan Fuller, Yiting Sun, Omar Chabklo, Andres Rodriguez, Jeman Park, and Brendan Saltaformaggio. 2022. Mistrust Plugins You Must: A {Large-Scale} Study Of Malicious Plugins In {WordPress} Marketplaces. In *31st USENIX Security Symposium (USENIX Security 22)*. 161–178.

[10] Seiya Kato, Rui Tanabe, Katsunari Yoshioka, and Tsutomu Matsumoto. 2021. Adaptive Observation of Emerging Cyber Attacks targeting Various IoT Devices. In *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. 143–151.

[11] Xigao Li, Babak Amin Azad, Amir Rahmati, and Nick Nikiforakis. 2021. Good bot, bad bot: Characterizing automated browsing activity. In *2021 IEEE symposium on security and privacy (sp)*. IEEE, 1589–1605.

[12] Xiaofeng Liu, Qixu Liu, Xiaoxi Wang, and Zhaopeng Jia. 2016. Fingerprinting Web Browser for Tracing Anonymous Web Attackers. In *2016 IEEE First International Conference on Data Science in Cyberspace (DSC)*. 222–229. doi:10.1109/DSC.2016.78

[13] Massimiliano Masi, Giovanni Paolo Sellitto, Helder Aranha, and Tanja Pavleska. 2023. Securing critical infrastructures with a cybersecurity digital twin. *Software and Systems Modeling* 22, 2 (April 2023), 689–707. doi:10.1007/s10270-022-01075-0

[14] MITRE Corporation. July 24, 2024. Threat Modeling with ATT&CK. https://ctid.mitre.org/projects/threat-modeling-with-attack/. Accessed: April 7, 2025.

[15] Gustavo Nascimento and Miguel Correia. 2011. Anomaly-based intrusion detection in software as a service. In *2011 IEEE/IFIP 41st International Conference on Dependable Systems and Networks Workshops (DSN-W)*. 19–24. doi:10.1109/DSNW.2011.5958858

[16] Maria Nintsiou, Elisavet Grigoriou, Paris Alexandros Karypidis, Theocharis Saoulidis, Eleftherios Fountoukidis, and Panagiotis Sarigiannidis. 2023. Threat intelligence using Digital Twin honeypots in Cybersecurity. In *2023 IEEE International Conference on Cyber Security and Resilience (CSR)*. 530–537. doi:10.1109/CSR57506.2023.10224997

[17] Mirna El Rajab, Li Yang, and Abdallah Shami. 2024. Enhancing Network Intrusion Detection: An AutoML Pipeline with Efficient Digital Twin Synchronization. (Jan. 2024). doi:10.36227/techrxiv.170475326.65758197/v1 Publisher: Institute of Electrical and Electronics Engineers (IEEE).

[18] Daniel Ramsbrock, Robin Berthier, and Michel Cukier. 2007. Profiling Attacker Behavior Following SSH Compromises. *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)* (2007), 119–124. https://api.semanticscholar.org/CorpusID:18673418

[19] Marcelo Rodríguez, Gustavo Betarte, and Daniel Calegari. 2021. A Process Mining-based approach for Attacker Profiling. *2021 IEEE URUCON* (2021), 425–429. https://api.semanticscholar.org/CorpusID:245527180

[20] Ulya Sabeel, Shahram Shah Heydari, Khalil El-Khatib, and Khalid Elgazzar. 2023. Unknown, atypical and polymorphic network intrusion detection: A systematic survey. *IEEE Transactions on Network and Service Management* 21, 1 (2023), 1190–1212.

[21] Takayuki Sasaki, Katsunari Yoshioka, and Tsutomu Matsumoto. 2023. Who are you? OSINT-based Profiling of Infrastructure Honeypot Visitors. In *2023 11th International Symposium on Digital Forensics and Security (ISDFS)*. 1–6. doi:10.1109/ISDFS58141.2023.10131856

[22] Methaq A. Shyaa, Noor Farizah Ibrahim, Zurinahni Zainol, Rosni Abdullah, Mohammed Anbar, and Laith Alzubaidi. 2024. Evolving cybersecurity frontiers: A comprehensive survey on concept drift and feature dynamics aware machine and deep learning in intrusion detection systems. *Engineering Applications of Artificial Intelligence* 137 (2024), 109143. doi:10.1016/j.engappai.2024.109143

[23] Methaq A Shyaa, Noor Farizah Ibrahim, Zurinahni Zainol, Rosni Abdullah, Mohammed Anbar, and Laith Alzubaidi. 2024. Evolving cybersecurity frontiers: A comprehensive survey on concept drift and feature dynamics aware machine and deep learning in intrusion detection systems. *Engineering Applications of Artificial Intelligence* 137 (2024), 109143.

[24] Oleksii Starov, Johannes Dahse, Syed Sharique Ahmad, Thorsten Holz, and Nick Nikiforakis. 2016. No Honor Among Thieves: A Large-Scale Analysis of Malicious Web Shells. In *Proceedings of the 25th International Conference on World Wide Web (WWW '16)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 1021–1032. doi:10.1145/2872427.2882992 event-place: Montréal, Québec, Canada.

[25] Samaneh Tajalizadehkhoob, Carlos Gañán, Arman Noroozian, and Michel van Eeten. 2017. The Role of Hosting Providers in Fighting Command and Control Infrastructure of Financial Malware. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security* (Abu Dhabi, United Arab Emirates) *(ASIA CCS '17)*. Association for Computing Machinery, New York, NY, USA, 575–586. doi:10.1145/3052973.3053023

[26] Bayu Adhi Tama, Lewis Nkenyereye, S.M. Riazul Islam, and Kyung-Sup Kwak. 2020. An Enhanced Anomaly Detection in Web Traffic Using a Stack of Classifier Ensemble. *IEEE Access* 8 (2020), 24120–24134. doi:10.1109/ACCESS.2020.2969428

[27] Seba Anna Varghese, Alireza Dehlaghi Ghadim, Ali Balador, Zahra Alimadadi, and Panos Papadimitratos. 2022. Digital Twin-based Intrusion Detection for Industrial Control Systems. In *2022 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*. 611–617. doi:10.1109/PerComWorkshops53856.2022.9767492

[28] Huan Wang, Xiaoqiang Di, Yan Wang, Bin Ren, Ge Gao, and Junyi Deng. 2023. An Intelligent Digital Twin Method Based on Spatio-Temporal Feature Fusion for IoT Attack Behavior Identification. *IEEE Journal on Selected Areas in Communications* 41, 11 (2023), 3561–3572. doi:10.1109/JSAC.2023.3310091

[29] Mengying Wu, Geng Hong, Jinsong Chen, Qi Liu, Shujun Tang, Youhao Li, Baojun Liu, Haixin Duan, and Min Yang. 2025. Revealing the black box of device search engine: scanning assets, strategies, and ethical consideration. In *Proceedings 2025 Network and Distributed System Security Symposium (NDSS 2025)*. Internet Society. doi:10.14722/ndss.2025.241924

[30] Jun Zhao, Qiben Yan, Xudong Liu, Bo Li, and Guangsheng Zuo. 2020. Cyber threat intelligence modeling based on heterogeneous graph convolutional network. In *23rd international symposium on research in attacks, intrusions and defenses (RAID 2020)*. 241–256.

Appendix

A Dataset Field Descriptions

Table 2 lists the structured fields available in the HTTP(S) dataset. The fields cover protocol metadata, HTTP header and payload contents, session timing, and IP related attributes for both source and destination.

Table 2: Structured Fields in the HTTP(S) Dataset

Field	Description
protocol	Communication protocol used (e.g., HTTP, HTTPS)
category	Labeled class of the session: intrusion, scouting, or scanning
id	Unique session identifier
http_method	HTTP method (GET, POST, etc.)
http_status	HTTP response status code
http_uri	Requested URI path
http_body	Payload/body content of the HTTP request
http_accept	Accept header field in HTTP request
http_accept_encoding	Encoding preferences (gzip, deflate, etc.)
http_user_agent	User-Agent string of the client
http_content_type	Content-Type header of the HTTP request
http_referer	Referer header from previous page
http_connection	Connection directive (e.g., keep-alive, close)
sessionTimeout	Boolean indicating if the session ended via timeout
sessionLength	Duration of the session (in seconds)
startTime	Session start timestamp
endTime	Session end timestamp
clientIP	Source IP address of the request
client_asn	Autonomous System Number of client IP
client_as_org	Organization name associated with client IP
client_country	Country associated with the client IP
hostIP	Destination IP address (sensor)
host_asn	ASN of the host (sensor)
host_as_org	Organization name for host IP
host_country	Country of the host sensor

B User-Agent Classification

Table 3 shows the categorization of user-agent strings based on common keyword matches. Each category groups similar client types according to tool names, libraries, or behavioral patterns observed in the user-agent field.

Table 3: User-Agent Category Classification

User-Agent Category	Representative Keywords
browser	mozilla, chrome, safari, edge, firefox, opera
cli tool	curl, wget, httpie
python lib	python, requests, aiohttp, urllib
scanner bot	expanse, paloalto, scanner, bot, censys, nmap, zgrab, modatscanner, wpdetector, internetmeasurement, genomecrawler
custom client	custom, asynchttpclient, fasthttp, l9explore, keydrop

C Feature Engineering Pipeline

This pipeline illustrated in Figure 9 extracts a comprehensive set of features from HTTP(S) traffic across six categories: basic protocol features, body and URI patterns, encoding types, spatial origin, and temporal behavior. Tokenization, embedding, and binary encoding techniques are used to transform both categorical and semantic inputs into a unified vector representation for downstream classification.

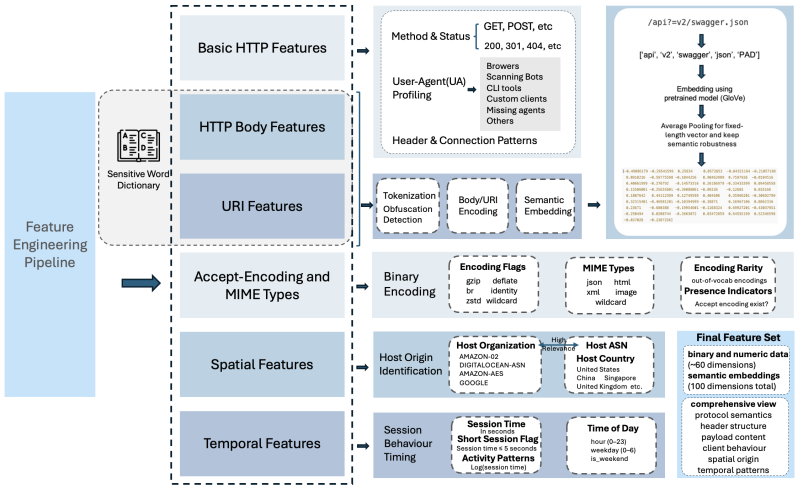


Figure 9: Overview of the feature engineering pipeline

D Fingerprint Feature Descriptions

Table 4 lists the 61 normalized features extracted from HTTP sessions for use in fingerprinting analysis. The features include categorical encodings, header indicators, content attributes, and URI-based embeddings.

Table 4: Normalized HTTP session features used in fingerprint histograms

Feature Name	Description
http_method	Encoded HTTP method (e.g., GET, POST)
http_status	Encoded HTTP response status code
body_has_exec_code	Binary indicator for executable code in body
body_length	Length of HTTP request body
accepts_gzip / deflate / br	Whether gzip/deflate/brotli is accepted
accepts_json / html / xml / image	MIME type preferences in Accept header
accepts_wildcard_mime	Accepts all MIME types ('*/*')
has_accept_encoding	Whether Accept-Encoding is explicitly set
conn_keep_alive / close / upgrade	Connection preferences (e.g., Keep-Alive)
has_connection_header	Presence of a Connection header
connection_field_length	Length of Connection header field
uri_emb_0 to uri_emb_49	50-dimensional URI embedding features

E Full Intrusion Taxonomy with Regex Matching

Table 5 outlines the full intrusion taxonomy used in the analysis, including category names, techniques, intended goals, and representative regex-based patterns. The taxonomy supports structured classification of observed intrusion behaviors based on HTTP session content and metadata.

Table 5: Full Intrusion Taxonomy with Regex-Based Pattern Examples

Intrusion Category	Technique	End Goal	Pattern Examples
Exploit Attempts	File Inclusion (LFI/RFI)	Code Execution	`?file=.*`, `?path=.*`, `?url=https?://.*`, `wp_abspath=`, `../`, `%2e%2e`
	Misconfiguration Exploit	Priv Esc / Info Leak	`.env`, `.bak`, `/setup.php`, `xmlrpc.php`, `/.git/`, `/config(.php)?`, `/backup/`, `/readme.html`, `/changelog.txt`
	REST/JSON Abuse	Data Leak / Enumeration	`/wp-json/`, `/api/`, `/wp-json/wp/v2/`, `/api/v[0-9]+/`
	SQL Injection (SQLi)	DB Access / Bypass	`select.+from`, `union.+select`, `or 1=1`, `--`, `%27`, `%3D`, `select`, `information_schema`
	Command Injection	Code Execution	`cmd=`, `system(`, `;\s*(wget|curl|bash)`, `&&`, `||`, `python -c`, `bash -i`, `nc -e`
	Denial of Service (DoS)	Resource Exhaustion	`xmlrpc.php`, `pingback.ping`, `/.well-known/`, `Content-Length: \d{7,}`, `slowloris`, `.zip`, `/flood`, `keep-alive`
Web Shell / Payload Upload	Simple Shell Upload	Persistent Access	`.php`, `cmd=`, `system(`, `shell.php`, `r57.php`, `c99.php`
	Obfuscated Shell Upload	Stealth Backdoor	`eval(base64_decode(`, `base64_decode(`, `gz(inflateuncompress)(|`, `str_rot13(`, `hex2bin(`
	Two-Stage Payload	Loader & Dropper	`upload`, `/shell.php?cmd=`, `wget http`, `curl -o`, `chmod +x`
Post-Exploitation Activity	Botnet C2 Callback	Remote Control	`/irc.php`, `/update`, `beacon`, `/gate.php`, `/bot/ping`, `/ping.php`
	Cronjob Deployment	Persistence	`@reboot`, `crontab`, `wget http`, `curl -o`, `chmod +x`, `bash .*&`
	Spam Mailer Setup	Email Abuse	`mailer.php`, `sendmail`, `smtp_host`, `mail(`, `email=spam`, `bulkmail`
	Proxy/Relay Deployment	Lateral Movement	`proxy.php`, `forward=`, `relay`, `remote_host`, `remote_port`
Delivery / Downloader	Direct Script Drop	Code Execution	`.sh`, `.exe`, `.php`, `wget https?://`, `curl https?://`
	Drive-by Download / JS	User Exploitation	`<iframe`, `eval(atob(`, `window.location`, `document.location`, `setTimeout(.*eval`, `base64,`
Obfuscated / Anomalous Behavior	Junk Payload Flood	Resource Exhaustion	`POST .+`, `garbled`, `payload.*spam`, `AAAAA+`, `[A-Z]{100,}`, `[0-9]{100,}`
	Unknown Pattern	Undiscovered Variant	`[a-zA-Z0-9+/=]{200,}`, `[a-f0-9]{32,}`, `entropy`, `anomaly`, `randomized`, `X-Obfuscated`, `X-Custom`

F HTTP(S) Traffic Labeling Criteria

Algorithm 1 defines the rule-based procedure used to assign labels to HTTP(S) sessions. The labeling logic distinguishes between confirmed intrusions, targeted access attempts, and generic scans based on content indicators, credential presence, and URI patterns.

Algorithm 1: Rule-Based Labeling Procedure for HTTP(S) Sessions

Input: HTTP(S) session record with fields: `protocol`, `hostName`, `httpRequests`, `credentials`
Output: Assigned label: `intrusion`, `attempt`, or `scan`
if `protocol` $\notin$ *{"http", "https"}* **then**
  return `Not Applicable`

if *the session contains command execution indicators in either the body or headers* **then**
  Matching keywords include: `sh`, `exe`, `shell`, `download`, `curl`, `wget`, `powershell`, `cmd`, `/bin/bash`, `/bin/sh` ;
  return `intrusion`

if *the session includes any of the following conditions* **then**
    `credentials` field is present; **and**
    Request URI contains substrings such as: `wp-login.php`, `phpmyadmin`, `login`, `admin`, `backup`, `config`
  and the session does not match any intrusion keyword in body or headers
return `attempt`

return `scan` ; `// Generic or exploratory HTTP traffic`

G Adaptive Intrusion Detection Workflow

Algorithm 2 describes the adaptive intrusion detection loop that monitors accuracy degradation and unseen patterns over time. It uses a dual-model setup (Random Forest and XGBoost) alongside a probabilistic Trie structure, updating models and dictionaries when accuracy drops or unknown sequence rates exceed predefined thresholds.

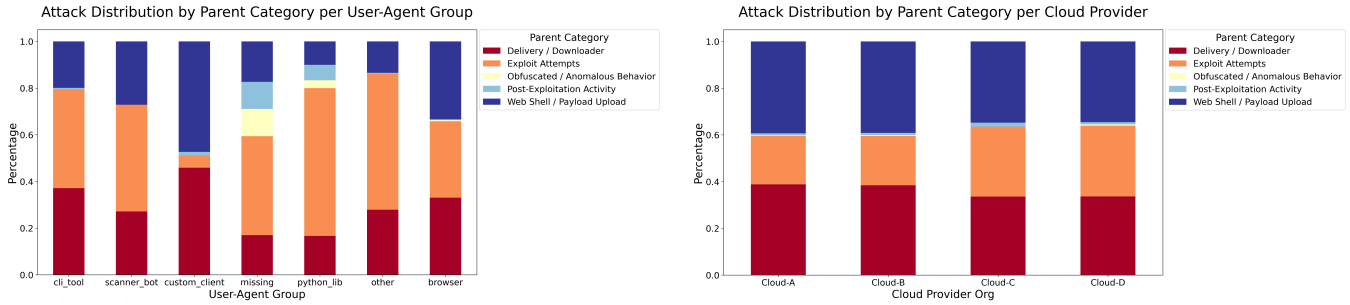
Algorithm 2: Adaptive Intrusion Detection with Dual-Model and Probabilistic Trie Update

Input: Start date S, End date E, Window size W, Accuracy threshold τ_{acc}, Unknown rate threshold τ_{unk}
Output: Evaluation log and updated models
Initialize training window $T \leftarrow [S, S + W - 1]$;
Train Random Forest and XGBoost models on window T ;
Extract sensitive words and build initial probabilistic Trie from T ;
Save baseline accuracies: $A^{\text{RF}}, A^{\text{XGB}}$;
foreach *day d from S to E* **do**
  Load daily data D_d ;
  Evaluate both models on D_d: $A_d^{\text{RF}}, A_d^{\text{XGB}} \leftarrow$ test accuracy ;
  Compute accuracy drops: $\Delta A^{\text{RF}} \leftarrow A^{\text{RF}} - A_d^{\text{RF}}$;
  $\Delta A^{\text{XGB}} \leftarrow A^{\text{XGB}} - A_d^{\text{XGB}}$;
  Check for new sensitive words and unknown sequences U_d via Trie matching ;
  Compute unknown rate: $R_{\text{unk}} \leftarrow \frac{|U_d|}{|D_d|}$;
  if $\Delta A^{RF} > \tau_{acc}$ ***and*** $\Delta A^{XGB} > \tau_{acc}$ ***or*** $R_{unk} > \tau_{unk}$ **then**
    Update training window: $T \leftarrow [d - W + 1, d]$;
    Retrain both models on new window T ;
    Update sensitive word dictionary and rebuild Trie from T ;
    Save updated models ;
  Log evaluation results for day d ;
Save full evaluation log to CSV ;

H Distribution Overview of Intrusion Categories

Stacked Distribution View. Figure 10 shows the proportional distributions of intrusion parent categories across user-agent groups and cloud providers. User-agent behaviors are more diverse, with varying emphases on exploit attempts or payload delivery, while cloud-based traffic skews uniformly toward delivery and post-exploitation. These patterns align with earlier fingerprinting results, highlighting greater behavioral variation from client tools than hosting infrastructure.

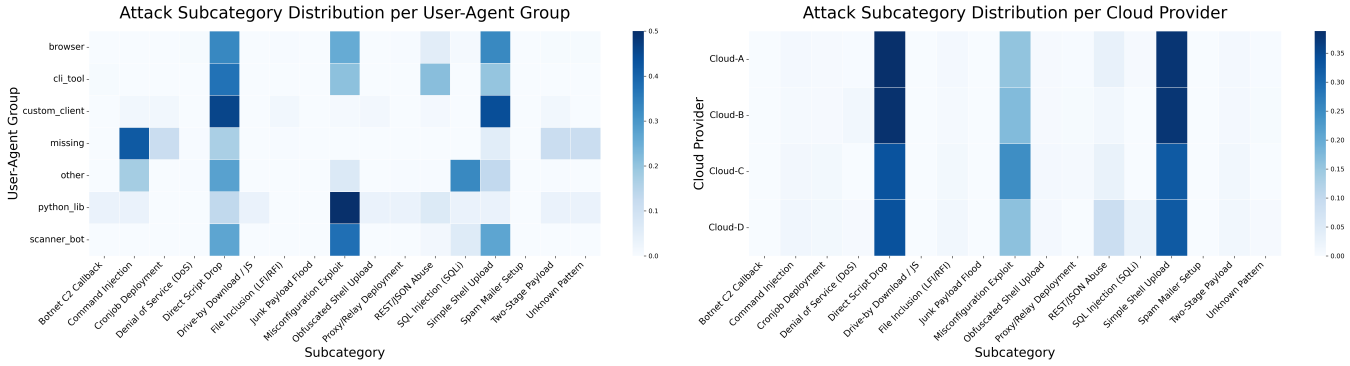
Heatmap View. Figure 11 presents normalized heatmaps of intrusion subcategories. User-agent groups show distinct profiles—cli tools and browsers favor exploit and delivery tactics, while python libraries and scanner bots target broader vectors like misconfigurations and



(a) Distribution of parent intrusion categories by user-agent group. **(b) Distribution of parent intrusion categories by cloud provider.**

Figure 10: Stacked percentage distributions of intrusion parent categories aligned with user-agent groups and cloud providers. These breakdowns validate the first level taxonomy structure used in fingerprinting and divergence analysis.

file inclusions. Cloud-based activity remains consistent, focusing on shell uploads and delivery. This further supports the conclusion that attacker behavior is shaped more by tooling than by infrastructure.



(a) Subcategory-level distribution across user-agent groups. **(b) Subcategory-level distribution across cloud providers.**

Figure 11: Normalized heatmaps of intrusion subcategory usage across user-agent and cloud groupings.

I Discovery of Unknown Sequences.

Figure 12 summarizes the total number of unknown sequences discovered under each sliding window configuration. Smaller windows (e.g., $w = 2$ to $w = 4$) detect more new patterns, reflecting higher sensitivity to localized variations. Larger windows ($w = 5$ to $w = 7$) yield fewer discoveries, indicating greater stability at the cost of reduced responsiveness.

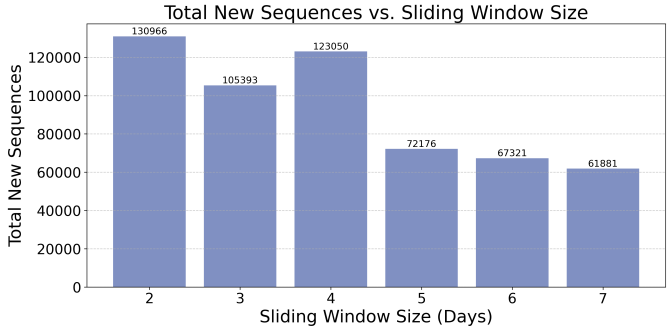


Figure 12: Total new sequence discoveries vs. adaptive sliding window size.

J Adaptive Retraining Evaluation

Table 6 summarizes the top 3 evaluation records for each training window size, based on test-day accuracy and Trie-based novelty metrics. For both Random Forest and XGBoost models, the table reports baseline, test accuracy, accuracy drop, and updates in sensitive word coverage and trie paths.

Table 6: Sliding Window Evaluation of Model Stability and Generalization – Top 3 records for each window

Window Size	Data Source		ML Accuracy (RF/XGB)			Trie Tree Measurement			Model Update
	Training Days	Test Day	Baseline Accuracy (%)	Test Day Accuracy (%)	Accuracy Drop (%)	New Sensitive Words	New Trie Paths Count	Unknown Rate (%)	
2	2025-03-15 to 16	2025-03-17	95.62/93.85	92.70/91.71	2.92/2.14	327	23,714	12.85	Yes
	2025-03-16 to 17	2025-03-18	95.39/93.81	90.56/89.46	4.83/4.34	88	5,693	3.53	No
	2025-03-16 to 17	2025-03-19	95.39/93.81	92.33/92.43	3.06/1.38	206	2,500	5.09	Yes
3	2025-03-15 to 17	2025-03-18	95.69/93.75	90.53/90.08	5.16/3.67	52	5,076	3.15	No
	2025-03-16 to 18	2025-03-19	95.61/93.65	93.17/92.77	2.44/0.88	182	1,555	3.16	No
	2025-03-17 to 19	2025-03-20	95.83/94.07	92.19/92.04	3.64/2.02	97	1,362	0.96	No
4	2025-03-15 to 18	2025-03-19	95.76/93.87	93.37/92.69	2.40/1.18	167	352	0.72	No
	2025-03-15 to 18	2025-03-20	95.76/93.87	92.12/91.75	3.65/2.13	97	32,064	22.54	Yes
	2025-03-17 to 20	2025-03-21	95.78/93.92	91.24/90.55	4.55/3.36	69	3,085	2.35	No
5	2025-03-15 to 19	2025-03-20	95.81/93.65	92.32/92.04	3.49/1.61	76	1,092	0.77	No
	2025-03-15 to 19	2025-03-21	95.81/93.65	91.12/90.74	4.69/2.91	122	2,897	2.21	No
	2025-03-15 to 19	2025-03-22	95.81/93.65	90.67/90.28	5.14/3.37	15	714	1.29	No
6	2025-03-15 to 20	2025-03-21	95.73/93.71	91.83/90.88	3.90/2.83	45	2,504	1.91	No
	2025-03-15 to 20	2025-03-22	95.73/93.71	90.78/90.46	4.95/3.26	10	661	1.19	No
	2025-03-15 to 20	2025-03-23	95.73/93.71	90.09/89.57	5.64/4.15	5	1,250	3.81	No
7	2025-03-15 to 21	2025-03-22	95.64/93.36	91.37/90.68	4.27/2.68	5	437	0.79	No
	2025-03-15 to 21	2025-03-23	95.64/93.36	90.60/89.60	5.04/3.76	3	436	1.33	No
	2025-03-15 to 21	2025-03-24	95.64/93.36	94.56/93.43	1.09/−0.07	0	46	0.28	No

Table 7 presents the detailed outcomes of adaptive retraining under a sliding window of $w = 6$, capturing how model performance and sequence generalization evolve over time. Each row tracks the system's response on a given test day, including accuracy degradation for both RF and XGB models, changes in sensitive keyword detection, and the emergence of new Trie paths. Integration with a new honeypot network (highlighted in gray) marks a shift in behavior, with spikes in unknown rate and sequence volume triggering retraining. These periods reflect higher variability in input patterns, underscoring the importance of dynamic model updating in maintaining detection effectiveness.

Table 7: Adaptive Retraining Evaluation Results ($w = 6$) – rows highlighted in gray indicate evaluation period with new honeypot integration.

Data Source		ML Accuracy (RF/XGB)			Trie Tree Measurement			Model Update
Training Days	Test Day	Baseline Accuracy (%)	Test Day Accuracy (%)	Accuracy Drop (%)	New Sensitive Words	New Trie Paths Count	Unknown Rate (%)	
2025-03-15 to 03-20	2025-03-15	95.73 / 93.71	99.25 / 94.20	-3.52 / -0.48	0	0	0.00	No
2025-03-15 to 03-20	2025-03-16	95.73 / 93.71	98.97 / 92.77	-3.24 / 0.94	0	1	0.00	No
2025-03-15 to 03-20	2025-03-17	95.73 / 93.71	99.13 / 93.78	-3.40 / -0.07	0	3	0.00	No
2025-03-15 to 03-20	2025-03-18	95.73 / 93.71	99.20 / 93.74	-3.47 / -0.02	0	0	0.00	No
2025-03-15 to 03-20	2025-03-19	95.73 / 93.71	99.13 / 94.50	-3.40 / -0.78	0	1	0.00	No
2025-03-15 to 03-20	2025-03-20	95.73 / 93.71	99.09 / 93.84	-3.36 / -0.12	0	0	0.00	No
2025-03-15 to 03-20	2025-03-21	95.73 / 93.71	91.83 / 90.88	3.90 / 2.83	2361	63	1.80	No
2025-03-15 to 03-20	2025-03-22	95.73 / 93.71	90.78 / 90.46	4.95 / 3.26	620	12	1.12	No
2025-03-15 to 03-20	2025-03-23	95.73 / 93.71	90.09 / 89.57	5.64 / 4.15	1249	7	3.81	Yes
2025-03-18 to 03-23	2025-03-24	95.33 / 93.31	94.24 / 93.87	1.09 / -0.57	69	0	0.42	No
2025-03-18 to 03-23	2025-03-25	95.33 / 93.31	90.03 / 86.27	5.30 / 7.04	1835	107	2.29	No
2025-03-18 to 03-23	2025-03-26	95.33 / 93.31	65.79 / 73.16	29.54 / 20.15	99373	1135	41.99	Yes
2025-03-21 to 03-26	2025-03-27	95.16 / 92.42	93.35 / 91.83	1.82 / 0.58	4733	93	1.60	No
2025-03-21 to 03-26	2025-03-28	95.16 / 92.42	92.63 / 89.76	2.54 / 2.66	20270	155	7.03	Yes
2025-03-23 to 03-28	2025-03-29	95.93 / 93.60	94.17 / 93.66	1.76 / -0.06	2719	124	1.03	No
2025-03-23 to 03-28	2025-03-30	95.93 / 93.60	93.02 / 90.90	2.91 / 2.69	3724	45	1.61	No
2025-03-23 to 03-28	2025-03-31	95.93 / 93.60	91.35 / 90.71	4.58 / 2.89	5738	84	3.00	Yes
2025-03-26 to 03-31	2025-04-01	95.85 / 93.82	87.52 / 86.26	8.33 / 7.56	976	20	1.82	Yes
2025-03-27 to 04-01	2025-04-02	95.77 / 93.94	89.34 / 87.61	6.44 / 6.32	142	3	0.82	Yes
2025-03-28 to 04-02	2025-04-03	95.71 / 93.98	91.84 / 90.44	3.86 / 3.54	8529	85	8.30	Yes
2025-03-29 to 04-03	2025-04-04	95.43 / 93.56	93.40 / 92.23	2.02 / 1.34	3037	46	3.03	Yes
2025-03-30 to 04-04	2025-04-05	95.44 / 93.49	90.66 / 88.46	4.77 / 5.03	2807	393	2.16	No
2025-03-30 to 04-04	2025-04-06	95.44 / 93.49	75.23 / 73.59	20.21 / 19.91	17682	350	14.69	Yes
2025-04-01 to 04-06	2025-04-07	95.83 / 92.94	84.85 / 93.12	10.98 / -0.18	1815	136	1.16	No
2025-04-01 to 04-06	2025-04-08	95.83 / 92.94	86.26 / 91.87	9.57 / 1.06	3211	168	1.99	No
2025-04-01 to 04-06	2025-04-09	95.83 / 92.94	85.60 / 89.58	10.24 / 3.36	6927	91	9.03	Yes