Domain Intelligence & Behaviour System (DIBs)

Version 1.0.1

A DNS Intelligence & Behaviour Analysis Framework

Technical Whitepaper

Author

Biswadeb Mukherjee

Offensive Security Specialist & Independent Researcher
ORCID: 0009-0005-4610-4010

DOI: 10.5281/zenodo.19432181

April 6, 2026
https://official-biswadeb941.in

Abstract

Domain registration ecosystems continue to be systematically abused as a foundational layer for phishing, malware distribution, spam campaigns, and botnet infrastructure. As of early 2026, daily registrations range between 250,000 and 400,000, with `.com` accounting for the largest share of malicious domains [1]. This paper presents findings from a live intelligence operation using DIBs — a purpose-built framework for high-throughput DNS resolution, domain mutation analysis, and infrastructure correlation. Starting from 500 Indian brand keywords, the system generated 330,576 candidate domains across six mutation techniques, of which 12,860 resolved. Candidates were filtered using Jaro-Winkler similarity and scored using a multifactor system based on entropy, lexical structure, and generation context.

Results reveal focused targeting of the Indian digital ecosystem, with `.com` dominating resolved infrastructure at 3,640 domains, followed by `.in` at 2,143 — reflecting both global registration trends and region-specific targeting. Brand-plus-function naming patterns and infrastructure concentration within AWS, Cloudflare, and GCP indicate automated provisioning and enable coordinated abuse reporting as a mitigation path.

Keywords: Domain Intelligence, DNS Behavioural Analysis, Threat Intelligence, Phishing Detection, Malicious Domain Analysis, Brand Impersonation, ASN Intelligence, Offensive Security

Executive Summary

Monitoring domain infrastructure at scale requires more than periodic spot checks. Threat actors register variations of legitimate domains, park lookalikes ahead of campaigns, and pivot across shared hosting infrastructure in ways that are difficult to detect unless the full mutation space is systematically generated and probed. DIBs solves this by treating domain generation, DNS resolution, and intelligence extraction as a single continuous pipeline rather than three separate tools. Operators provide seed keywords — brand names, product strings, organisation identifiers — and the system derives the complete mutation space, resolves each candidate, and produces enriched NDJSON telemetry that maps domains to IPs, ASNs, providers, nameservers, and registration metadata. In the research run documented in this paper, 500 Indian brand keywords were used as the seed corpus.

The design makes three deliberate engineering choices:

- **Durability over convenience.** Generated domains are persisted to a SQLite spool before resolution begins. If the process is interrupted, it resumes without re-generating and without re-resolving domains already seen. The resolve interval is configurable, so the same spool can be re-used across recurring scan cycles.
- **Rate control by design.** A token-bucket rate limiter backed by Redis, combined with a cooldown gate that can be triggered automatically or by operator signal, ensures the framework does not generate excessive query volume against upstream resolvers or trigger automated abuse responses.
- **Structured output, no proprietary schema.** Every output is NDJSON. There is no built-in dashboard or storage layer. The pipeline ends at files on disk, which can be ingested by any tool the operator already has.

The results from the initial research run, documented in this paper, confirm that the pipeline operates at the intended throughput with predictable resource consumption and produces intelligence records that are immediately usable in ASN clustering and provider attribution workflows.

Contents

1 Introduction

1.1 Problem Statement

DNS forms a foundational layer of internet infrastructure, translating human-readable domain names into IP addresses required for network communication [2, 3, 4]. As a globally distributed system, it serves as a critical observation point where malicious infrastructure often leaves detectable traces. Passive DNS logs are useful for retroactive analysis, but they are not generated if no one queries the domain. Domain monitoring services watch a fixed list of registered domains, but they do not probe the unregistered space ahead of threat actor activity. Offensive security operators face a similar gap from the opposite direction. During red team engagements or threat intelligence work, understanding which lookalike domains are live, which resolve to cloud infrastructure, and which share IP prefixes with known malicious hosts requires tooling that can generate, resolve, and correlate at scale. Hand-crafting domain lists is slow and incomplete. DIBs was built to fill this gap. It is not a passive monitoring system and it is not a threat feed. It is an active generation and resolution engine whose job is to produce a comprehensive, enriched picture of the domain space surrounding a set of seed keywords.

The DIBs framework, including its full implementation and execution artifacts, is publicly available [5].

1.2 Scope and Intended Audience

This paper is written for engineers who need to understand how the system works at the implementation level, not for a marketing audience. The reader is assumed to be comfortable with Go, DNS protocol mechanics, and basic distributed systems concepts. The system is designed for:

- Threat intelligence operations requiring systematic domain space enumeration
- Red team reconnaissance against a target's brand or product namespace
- Phishing infrastructure detection through proactive resolution of lookalike domains
- DNS telemetry research where a reproducible, structured dataset is required
- Incident response cases where shared infrastructure correlation is needed at speed

1.3 Design Goals

The design constraints that shaped this system were established at the start of the project and held through implementation:

- **Completeness over speed.** The mutation engine must cover the full known-effective technique space — bitsquatting, typosquatting, combosquatting, homograph, subdomain, and phonetic — not just the quick wins. Filtered by Jaro-Winkler similarity to keep results relevant.
- **Resumability.** Long-running scans must survive interruption without re-processing completed work. This required a persistent spool from the beginning.

- **Rate discipline.** Generating 330,000+ domains and resolving them has the potential to look like an attack from the perspective of upstream resolvers. The rate limiter and cooldown gate are first-class components, not afterthoughts.
- **Operator control.** The HTTP API exists so the operator can stop, query status, and pull metrics mid-run without touching the process directly.
- **No storage opinion.** The system writes NDJSON to disk. What happens to those files is the operator's problem. No embedded database, no web UI, no retention policy.

1.4 Technology Choices

Go was chosen for its concurrency primitives, predictable memory behaviour, and the quality of the DNS library ecosystem (miekg/dns). Redis was chosen for rate limiting and the domain work queue because its atomic operations give the correct semantics for the sliding window token bucket and for BLPOP-based work distribution. SQLite was chosen for the generation spool because the access pattern is sequentially written followed by sequential reads — no need for a server-mode database.

2 Architecture

2.1 System Overview

DIBs is structured as a unidirectional processing pipeline with a control plane sitting alongside it. The pipeline has four major stages:

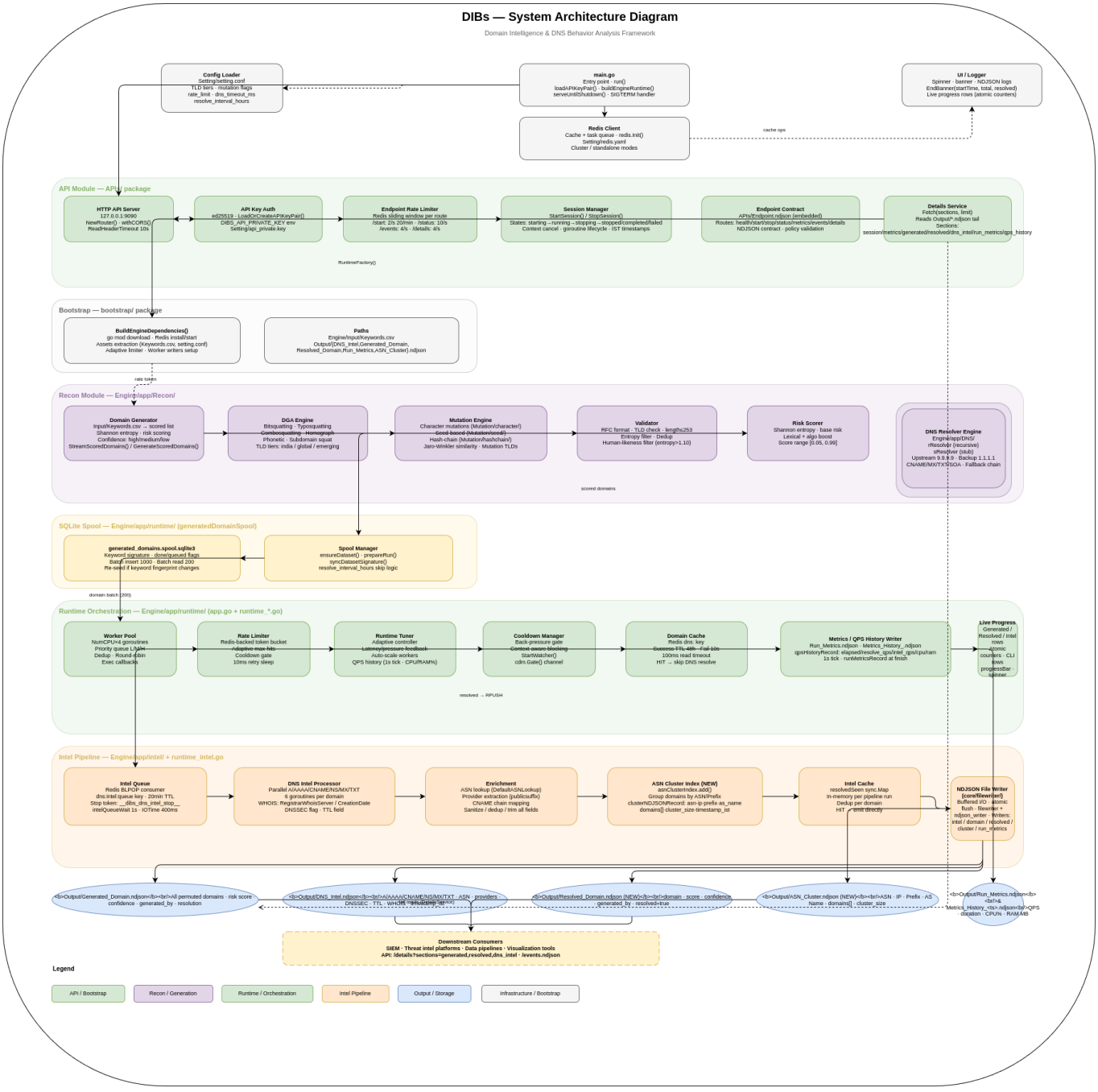

Figure 2.1.1: DIBs System Architecture Diagram — component relationships and data flow across all packages

The pipeline is divided into four primary stages: The control plane is a lightweight HTTP server (127.0.0.1:9090) that accepts start/stop/status/metrics requests authenticated by ed25519 public key. It does not participate in the data pipeline — it controls the goroutine lifecycle around it.

2.2 Data Flow — Bootstrap to DNS Resolution - Part 1

Part 1 of the data flow traces the path from the API request, through keyword loading and domain generation, into the SQLite spool, and finally into the DNS resolution stage with rate control.

2.2.1 Initialization Layer (P0a–P0e)

The following components execute once at startup and establish the runtime environment:

- **P0a**: HTTP server
- **P0b**: API key authentication
- **P0c**: Rate limiter
- **P0d**: Session manager
- **P0e**: Bootstrap initialization

These components collectively prepare the system for controlled execution and request handling.

2.2.2 Generation Pipeline (P1–P5b)

The generation pipeline is responsible for producing candidate domains:

- CSV reader ingests keywords from `Keywords.csv`
- Keywords are processed in parallel by the DGA engine and mutation engine
- Generated candidates are merged into a deduplicating collector
- Domains are validated and risk-scored
- Human-likeness filtering is applied
- Final candidates are written into the SQLite spool

This stage ensures high-quality, non-redundant domain generation before resolution.

2.2.3 Spool Management (P6)

The spool manager maintains consistency and avoids unnecessary recomputation:

- Uses **SHA-256 keyword fingerprinting** to detect changes in the input set
- If the keyword set remains unchanged and the resolve interval has not elapsed, the existing spool is reused
- Otherwise, the spool is regenerated

This mechanism optimizes performance and reduces redundant processing.

2.2.4 Resolution Loop (P7–P12)

The resolution loop processes domains from the spool through controlled DNS resolution:

- Cache check to avoid duplicate resolutions
- Task submission to the worker pool
- Cooldown gate to prevent rapid reprocessing
- Redis-based token bucket rate limiter
- DNS resolution execution
- Cache write for resolved entries

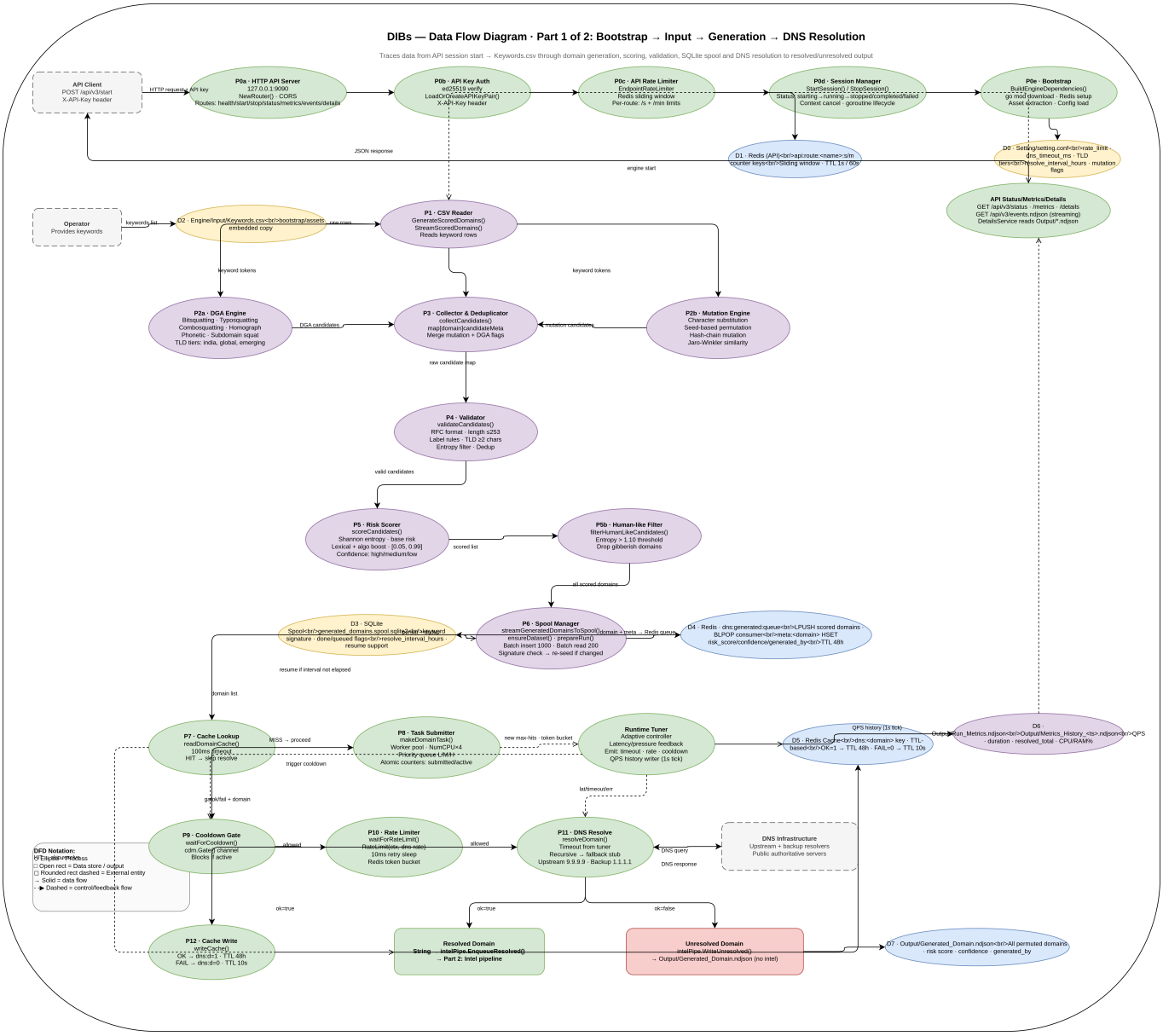

Figure 2.2.1: Data Flow Diagram Part 1 — Bootstrap → Input → Generation → DNS Resolution

Resolved domains are forwarded to the intelligence pipeline, while unresolved domains are written directly to the output.

2.3 Data Flow — Intel Pipeline to Output

Part 2 traces the path from a resolved domain through the intelligence extraction pipeline to the five output files. This stage runs concurrently with resolution and does not block the resolver.

- **P12–P14:** Redis intel queue consumer using `BLPOP`, with parallel DNS lookup goroutines for `A`, `AAAA`, `CNAME`, `MX`, `NS`, and `TXT` records, followed by WHOIS and ASN enrichment.
- **P14b:** ASN lookup against WHOIS infrastructure. Returns ASN number, IP prefix, and AS name per resolved IP address.
- **P15–P17:** Provider extraction from nameserver patterns, sanitisation, deduplication, and in-memory intel cache write with per-pipeline-run deduplication.
- **P17b:** ASN cluster index maintained in memory, accumulating domain-to-ASN/IP associations. Cluster records are emitted as soon as two or more domains share the same IP or ASN key.
- **P18–P19d:** Result router writes to four parallel NDJSON writers: intel, generated domain, cluster, and resolved domain records.

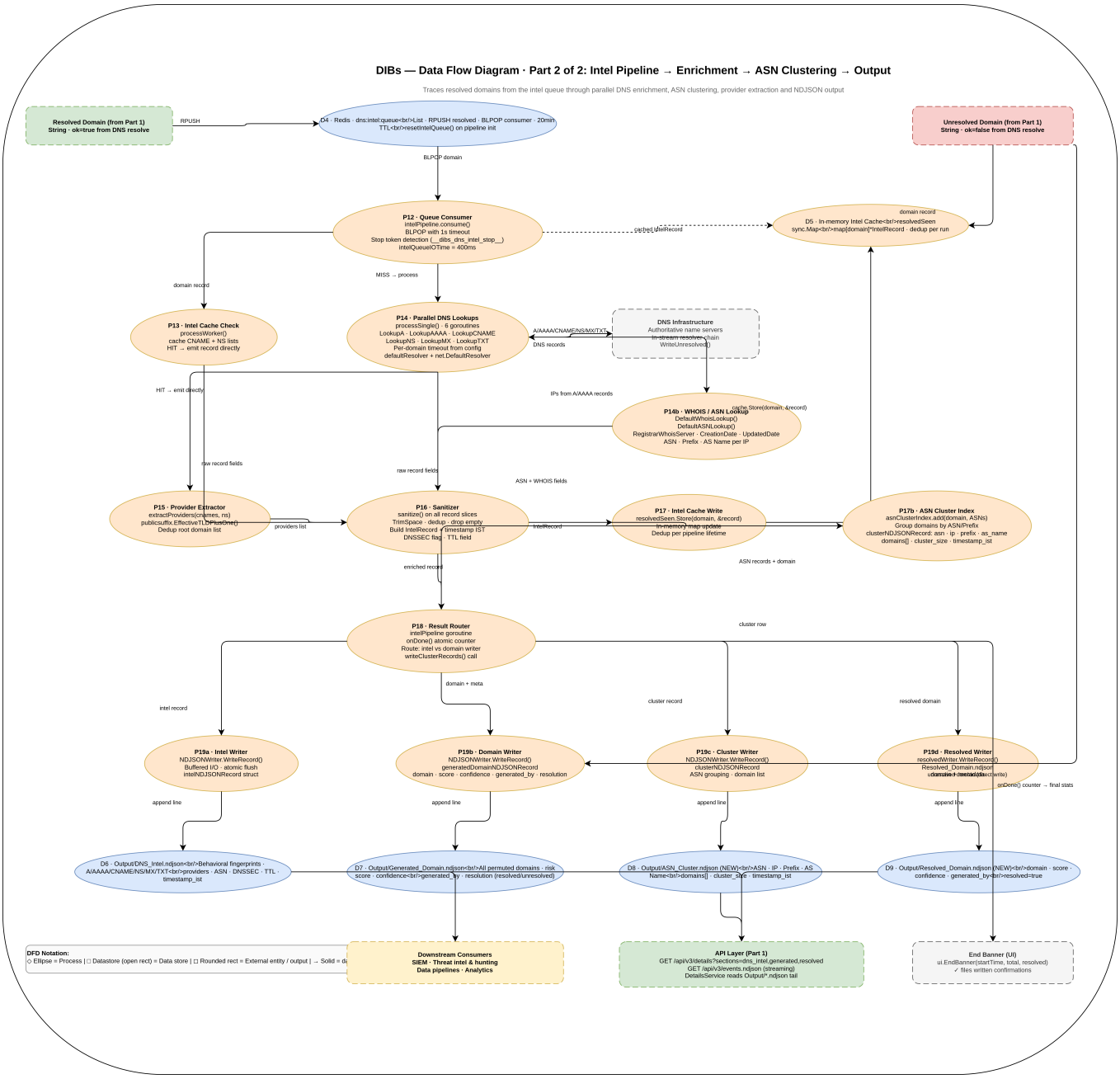

Figure 2.3.1: Data Flow Diagram Part 2 — Intel Pipeline → Enrichment → ASN Clustering → Output

3 Implementation

3.1 Domain Generation Engine

The generation engine is located in `Engine/app/Recon/`. It reads seed keywords from a CSV file, applies six mutation algorithms, appends TLD combinations from a configurable tiered TLD set, filters candidates using Jaro-Winkler similarity, assigns risk scores, and forwards them to the spool manager. All of this occurs before any DNS query is issued.

3.1.1 Mutation Algorithms

Six techniques are applied to each seed keyword:

- **Bitsquatting** — Flips individual bits in the ASCII encoding of each character. For a character at position i, each of the 8 bit positions is flipped in turn, and the resulting byte is validated against the allowed domain label character set (`a-z`, `0-9`, hyphen). Only safe substitutions are retained. This models both hardware memory error scenarios and deliberate bit-flip registrations.
- **Typosquatting** — Applies keyboard-adjacency transpositions, character deletions, insertions, and substitutions based on QWERTY layout proximity.
- **Combosquatting** — Prepends and appends commonly abused terms (e.g., `secure`, `login`, `verify`, `support`, `account`) to the base keyword.
- **Homograph Generation** — Substitutes visually similar Unicode characters for ASCII equivalents, with consideration for IDN encoding constraints.
- **Subdomain Squatting** — Constructs variations such as `keyword.tld`, `www-keyword.tld`, `mail.keyword.tld`, and similar patterns.
- **Sound Squatting** — Applies phonetic substitutions based on pronunciation equivalence (e.g., `ph` $\rightarrow$ `f`, `ck` $\rightarrow$ `k`).

3.1.2 Similarity Filtering

All generated candidates are filtered using the Jaro-Winkler distance metric, a widely adopted technique in record linkage and string similarity analysis for identifying closely related textual variations [6, 7].

3.1.3 TLD Tiers

The TLD set is divided into three configurable tiers defined in `setting.conf`:

- **Tier 1 — ccTLD family**

```
target_tlds_india = .in, .co.in, .net.in, .org.in, .in.net
```

- **Tier 2 — high-abuse gTLDs**

```
target_tlds_global = .com, .net, .org, .top, .xyz, .online,
.shop, .info, .vip, .sbs, .icu, .buzz,
.monster, .cfd, .lat
```

- **Tier 3 — opt-in**

```
target_tlds_emerging = .zip, .mov, .li, .beauty, .pics, .lol,
.finance, .support
```

The active tiers for the current run are defined as:

```
active_tld_tiers = india, global
```

3.1.4 SQLite Spool

Generated domains are written to a SQLite database (`generated_domains.spool.sqlite3`) co-located with the `Keywords.csv` input. The spool manager computes a SHA-256 fingerprint of the keyword file at startup. If the fingerprint matches the stored signature and the `resolve_interval_hours` window has not elapsed, the existing spool is reused instead of re-generating.

Batch insert size is 1,000 rows, while the consumer reads in batches of 200.

This design ensures that any interrupted run — due to process crash, operator `SIGTERM`, or power loss — can resume without re-generating domains or re-resolving already processed entries. The spool is fully populated before consumption begins; there is no streaming generation into resolution.

3.1.5 Risk Scoring

Each candidate is assigned a risk score derived in part from Shannon entropy, a fundamental measure of information randomness commonly applied in anomaly detection and pattern analysis [8].

Bitsquatting and typosquatting receive higher boosts compared to combosquatting. A human-likeness filter (entropy < 1.10 threshold) is applied to eliminate gibberish strings that may satisfy mutation rules but are unlikely to be registered by real-world actors.

3.2 DNS Resolution Engine

The DNS engine is located in `Engine/app/DNS/`. DNS resolution is performed using a Go-based implementation built on established libraries to ensure protocol-compliant query handling, retries, and timeouts [3, 9]. It wraps the `miekg/dns` library and supports two resolution modes: stub resolution against configured upstream resolvers, and recursive resolution that walks the delegation chain from the root.

The default mode for domain existence checking is stub resolution, as it provides sufficient accuracy with significantly better performance.

Configuration parameters from `setting.conf`:

```
upstream_dns = 9.9.9.9:53
backup_dns = 1.1.1.1:53
dns_retries = 2
dns_timeout_ms = 1500
```

The resolution workflow for a given domain proceeds as follows:

1. The domain is checked against the Redis cache. If a valid result exists within the TTL window (48 hours for resolved entries, 10 seconds for failed attempts), the cached result is returned immediately without issuing a DNS query.

2. The cooldown gate is evaluated. If a cooldown is active, the goroutine blocks on a channel until the gate reopens.
3. The rate limiter is consulted. If the token bucket is exhausted, the goroutine sleeps for 10 ms before retrying.
4. The DNS query is issued using the configured timeout and retry parameters.
5. The result is written back to the cache, and the domain is routed either to the intelligence pipeline queue or to the unresolved output.

3.3 Intelligence Extraction

Intelligence extraction operates within a separate goroutine consumer pool (`intelPipeline`) that reads from a Redis `BLPOP` queue populated by the resolver. This stage executes concurrently with DNS resolution and does not block resolver goroutines.

Processing for a single domain includes:

- **WHOIS and RDAP Enrichment** — WHOIS and RDAP services are used to enrich domain records with registration metadata, supporting attribution and temporal analysis [10]. Additionally, DNSSEC-related fields are considered to assess the presence of cryptographic validation mechanisms in DNS responses [11, 12, 13, 14].
- **Parallel DNS lookups** — Six goroutines per domain handle record types A, AAAA, CNAME, MX, NS, and TXT. Resolution uses `net.DefaultResolver` with a 400 ms I/O timeout per operation.
- **WHOIS enrichment** — Fields such as `RegistrarWhoisServer`, `CreationDate`, and `UpdatedDate` are retrieved via `DefaultWhoisLookup`.
- **ASN lookup** — Each IP obtained from A/AAAA records is mapped to its autonomous system number, IP prefix, and AS name. Results are stored per IP to support clustering.
- **Provider extraction** — Nameserver patterns and effective TLDs are matched against a known provider list to label hosting or DNS providers (e.g., Cloudflare, AWS, Azure, GCP).
- **DNSSEC and TTL extraction**
- **Sanitisation** — Empty fields are trimmed, CNAME and NS records are deduplicated, and timestamps are normalised to IST.

A per-run in-memory cache (`resolvedSeen sync.Map`) prevents duplicate processing. If a resolved domain has already been processed during the current run, results are served directly from cache without repeating lookups.

3.4 Correlation and Clustering

The ASN cluster index (`asnClusterIndex`) is maintained in memory throughout execution. Each finalised intelligence record contributes its domain and ASN/IP tuples to the index.

The index key is constructed as:

`AS12345|192.0.2.1`

A cluster record is emitted as a `clusterNDJSONRecord` when two or more distinct domains share the same ASN/IP combination.

Each record contains:

- ASN
- IP address
- IP prefix
- AS name
- List of co-hosted domains
- Cluster size

Cluster records are updated incrementally as new domains are added. The output writer receives continuous updates, so the final `ASN_Cluster.ndjson` file reflects the full evolution of cluster formation over time.

This design enables near-real-time detection of shared infrastructure. If a newly resolved domain joins a cluster containing a known malicious domain, the association is immediately reflected in the output without requiring the run to complete.

3.5 Runtime Orchestration

The runtime package (`Engine/app/runtime/`) is the largest component in the codebase. It manages the worker pool, rate limiter, cooldown manager, adaptive controller, progress display, and metrics writer.

3.5.1 Worker Pool

Workers are spawned at `runtime.NumCPU * 4`. Each worker operates as a goroutine consuming tasks from a priority queue (Low/Medium/High). Domain resolution tasks are submitted at Medium priority.

The worker pool supports:

- Task deduplication
- Round-robin dispatch
- Execution callbacks
- A monitor goroutine tracking active and completed task counts

3.5.2 Rate Limiter

The rate limiter is implemented as a Redis-backed sliding window token bucket. The initial rate is seeded from configuration (default 150 requests per second, ceiling 180 requests per second) and dynamically tuned based on total domain count and worker capacity.

A Lua `EVAL` script performs atomic check-and-decrement operations in Redis, preventing race conditions within the sliding window.

3.5.3 Cooldown Manager

The cooldown gate can be triggered automatically after a configurable number of completed domains (default: every 10,000 domains) or manually via an API.

When active, all worker goroutines blocked on the rate limiter also check a cooldown channel before proceeding. The default cooldown duration is 60 seconds and is configurable.

3.5.4 Adaptive Controller

The adaptive controller (`core/adaptive/`) implements a control loop that evaluates system pressure using metrics such as queue depth, in-flight task count, active workers, and completion rate.

It is capable of recommending adjustments to rate limits and timeout values. In the current implementation, it serves as infrastructure for future auto-scaling. The `autoscale` flag is disabled by default, and the tuner hooks (`observeTask`, `observeLimiterError`) are currently no-ops.

The controller itself is fully implemented and tested; integration with the runtime tuner is the next development step.

3.5.5 Progress and Metrics

A live progress display updates three rows in the terminal:

- Generated domain count
- Resolution progress (including cooldown state)
- Intelligence pipeline progress

All values are derived from atomic `int64` counters shared across goroutines.

A QPS history writer samples the following metrics every second:

- Resolve QPS
- Intelligence QPS
- CPU usage percentage
- Memory usage (MB)

Each sample is appended as a `qpsHistoryRecord` to a timestamped NDJSON file.

Additionally, a run metrics file captures overall execution duration, total generated domains, resolved domain count, and final rate limit values.

3.6 API Control Plane

The HTTP API listens on `127.0.0.1:9090` and is not intended for external exposure.

Authentication is implemented using `ed25519` public key validation via the `X-API-Key` header. The key pair is generated at first startup and stored in `Setting/api_private.key`. The corresponding public key is printed to `stdout` on each startup.

Endpoint behaviour is defined in `APIs/Endpoint.ndjson`, a machine-readable contract file validated by the router during initialization.

3.6.1 Endpoints

Method & Path	Auth	Notes
GET `/healthz`	No	Liveness probe
POST `/api/v3/start`	Yes	Starts a scan session
POST `/api/v3/stop`	Yes	Stop running session
GET `/api/v3/status`	Yes	Session state + timestamps
GET `/api/v3/metrics`	Yes	Runtime counters + QPS
GET `/api/v3/events`	Yes	Streams NDJSON tail

3.6.2 Rate Limiting

Per-route rate limits are defined in the contract:

- State-changing endpoints: 5 requests per second, 15 requests per minute
- Details/events endpoint: 4 requests per second

4 Results

The following data is from a single research run executed on 2 April 2026. The system ran unattended from **08:52:57** to **20:18:17 IST**, with a total execution time of **11 hours, 25 minutes, and 20 seconds**.

4.1 Run Summary

Metric	Value	Description
Total Domains Generated	330,576	Full mutation output across all techniques and TLD tiers
Total Domains Resolved	12,860	Successfully resolved domains (active infrastructure)
Total Domains Unresolved	317,716	Non-resolving domains (expected in mutation-heavy generation)
Resolution Rate	3.9%	Indicates proportion of active domains in generated space
Run Duration	11h 25m 20s	Continuous execution without operator intervention
Average Resolve QPS	∼15–25	Stable throughput with minor rate-control stalls
DNSSEC Observed	0.00%	No DNSSEC-enabled domains observed in dataset
Top DNS Provider	afternic.com	2,176 domains (4,087 combined with domaincontrol.com, representing 31.8% of all resolved domains)
Top Geo Location	United States	458 IPs (highest concentration of resolved infrastructure)
Largest ASN Cluster	AWS	∼ approximately 5,400 domains indicating large-scale cloud concentration

4.2 Domain Generation and Resolution

The domain generation pipeline produced a total of **330,576** candidate domains across all mutation techniques and active TLD tiers. The distribution of generated domains is heavily skewed toward techniques with higher combinatorial expansion.

Bitsquatting dominated the output with **173,320 domains (52.4%)**, followed by **typosquatting** at **68,384 (20.7%)**. Other techniques contributed smaller but still meaningful portions of the mutation space, including **subdomain squatting** at **26,239 (7.9%)**, **seed-based mutations** at **21,952 (6.6%)**, **character substitutions** at **15,785 (4.8%)**, and **combined character-plus-typo mutations** at **9,484 (2.9%)**.

This distribution is expected given the mechanics of each technique. Bitsquatting, in particular, generates up to eight variations per character position, leading to exponential growth for longer keywords. In contrast, techniques such as seed-based and character substitutions produce more constrained and semantically

meaningful variations.

From the full candidate set, **12,860 domains successfully resolved**, yielding a **3.9% resolution rate**. This is consistent with expectations for mutation-driven domain generation, where the majority of generated candidates remain unregistered. The low resolution rate should not be interpreted as inefficiency; rather, it reflects the exhaustive nature of the mutation space.

The operational value lies in the subset of domains that do resolve. These resolved domains represent active infrastructure and form the basis for downstream intelligence analysis. Notably, the resolved set maps to a significantly smaller number of hosting providers and autonomous systems, indicating infrastructure concentration and potential automation in domain provisioning.

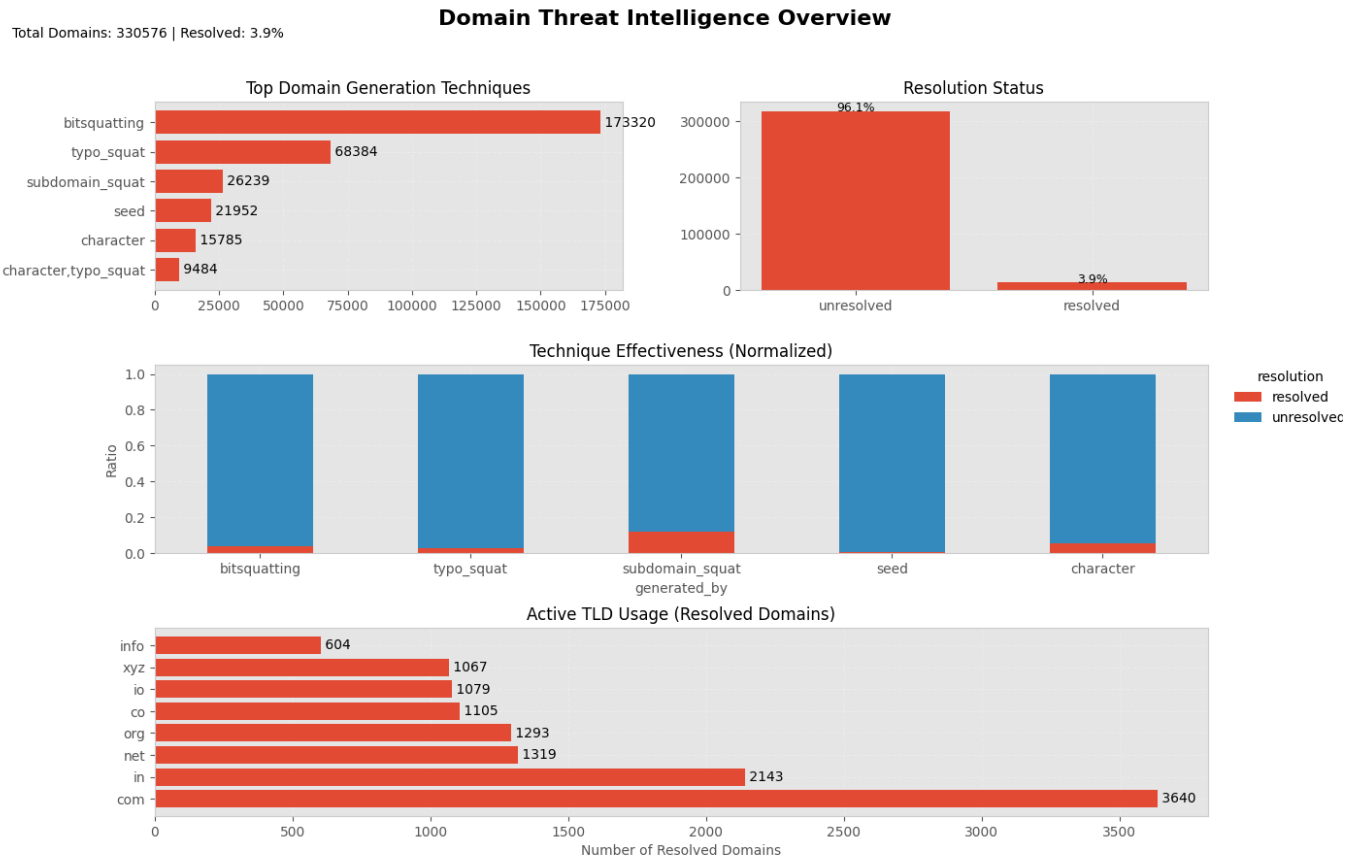

Figure 4.2.1: Domain Threat Intelligence Overview — generation distribution, resolution status, technique effectiveness, and TLD usage patterns

Figure 4.2.1 illustrates four key dimensions of the dataset. First, the generation distribution confirms the dominance of bitsquatting and typosquatting in terms of raw volume. Second, the resolution status highlights the imbalance between unresolved and active domains, reinforcing the need for large-scale enumeration to surface meaningful signals.

When normalized by resolution rate, **subdomain squatting exhibits a higher effectiveness ratio** compared to bitsquatting and typosquatting. This suggests that domains constructed using prefix-based or contextual variations (e.g., `login-`, `secure-`, `mail-`) are more likely to be actively registered. These patterns align with real-world phishing infrastructure, where attackers prioritize readability and user trust over minimal

perturbations.

In contrast, bit-flip and single-character variations, while abundant, show lower activation likelihood. These techniques are more effective for broad enumeration and detection coverage rather than precise targeting of active threats.

The analysis of active TLD usage among resolved domains further reinforces expected patterns. The **.com TLD dominates** with **3,640 resolved domains**, followed by **.in (2,143)**, **.net (1,319)**, **.org (1,293)**, **.co (1,105)**, **.io (1,079)**, **.xyz (1,067)**, and **.info (604)**.

This distribution reflects both global registration trends and region-specific targeting. The prominence of **.in** indicates a focused alignment with the Indian digital ecosystem, while high-abuse generic TLDs such as **.xyz**, **.info**, and **.online** remain consistently represented in active infrastructure.

Overall, the results demonstrate that while the majority of generated domains remain inactive, the resolved subset provides high-value intelligence. The combination of large-scale mutation, controlled resolution, and structured enrichment enables the identification of infrastructure clusters and attack patterns that would not be visible through passive observation alone.

4.3 DNS Infrastructure Analysis

A total of **12,860 resolved domains** were processed through the intelligence pipeline, with enrichment covering DNS records, nameserver patterns, and provider attribution. The resulting dataset provides a clear view into the infrastructure landscape supporting the active portion of the generated domain space.

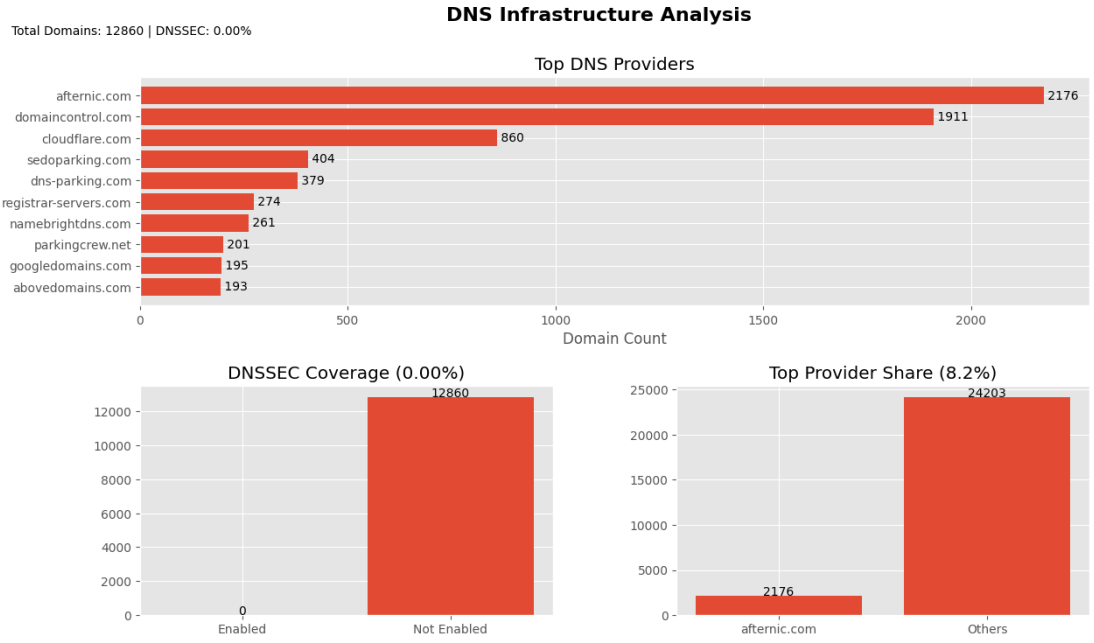

Figure 4.3.1: DNS Infrastructure Analysis — provider distribution, parking prevalence, and infrastructure concentration across resolved domains

The most prominent provider observed is **afternic.com**, accounting for **2,176 domains (8.2%)**. Afternic, operated by GoDaddy, is a domain parking and monetisation platform. When combined with **domaincon-**

trol.com (GoDaddy's primary authoritative nameserver infrastructure), the total rises to **4,087 domains**, representing approximately **31.8%** of all resolved domains.

This concentration is significant from a threat intelligence perspective. Parked domains are not inherently malicious, but they represent *pre-positioned infrastructure* that can be activated rapidly. The presence of a large parked domain base within mutation-derived candidates suggests that attackers may leverage registrar ecosystems to stage domains prior to campaign deployment.

Cloudflare accounts for **860 domains (6.7%)**, indicating a meaningful portion of actively managed infrastructure. Unlike parking services, Cloudflare-backed domains are more likely to be operational, benefiting from CDN masking, DNS abstraction, and ease of deployment. This aligns with common attacker behaviour where resilient and easily configurable infrastructure is preferred.

Additional parking providers contribute further to the resolved domain space, including **sedoparking.com (404 domains)**, **dns-parking.com (379 domains)**, and **parkingcrew.net (201 domains)**. When aggregated, these services represent a substantial share of the observed infrastructure, reinforcing the conclusion that domain parking plays a central role in the lifecycle of lookalike domains.

A notable observation is the complete absence of DNSSEC across all resolved domains — none of the 12,860 resolved domains had DNSSEC enabled, reflecting the generally low adoption of DNSSEC across the broader domain ecosystem [14]. While expected within a mutation-generated space dominated by parked and low-trust domains, it establishes a clear baseline and aligns with their transient or opportunistic usage patterns.

Overall, the infrastructure analysis reveals a dual pattern: a large proportion of domains remain parked within registrar-controlled ecosystems, while a smaller but critical subset is deployed on active infrastructure providers. This split highlights the importance of monitoring both inactive and active domains, as the transition between the two states can occur rapidly and without prior indication.

4.4 ASN and Geo-Intelligence

Each resolved domain was enriched through the WHOIS and ASN lookup pipeline, mapping IP addresses to their corresponding autonomous systems, IP prefixes, and geographic attribution. The resulting dataset was clustered based on shared ASN and IP relationships to identify infrastructure concentration patterns.

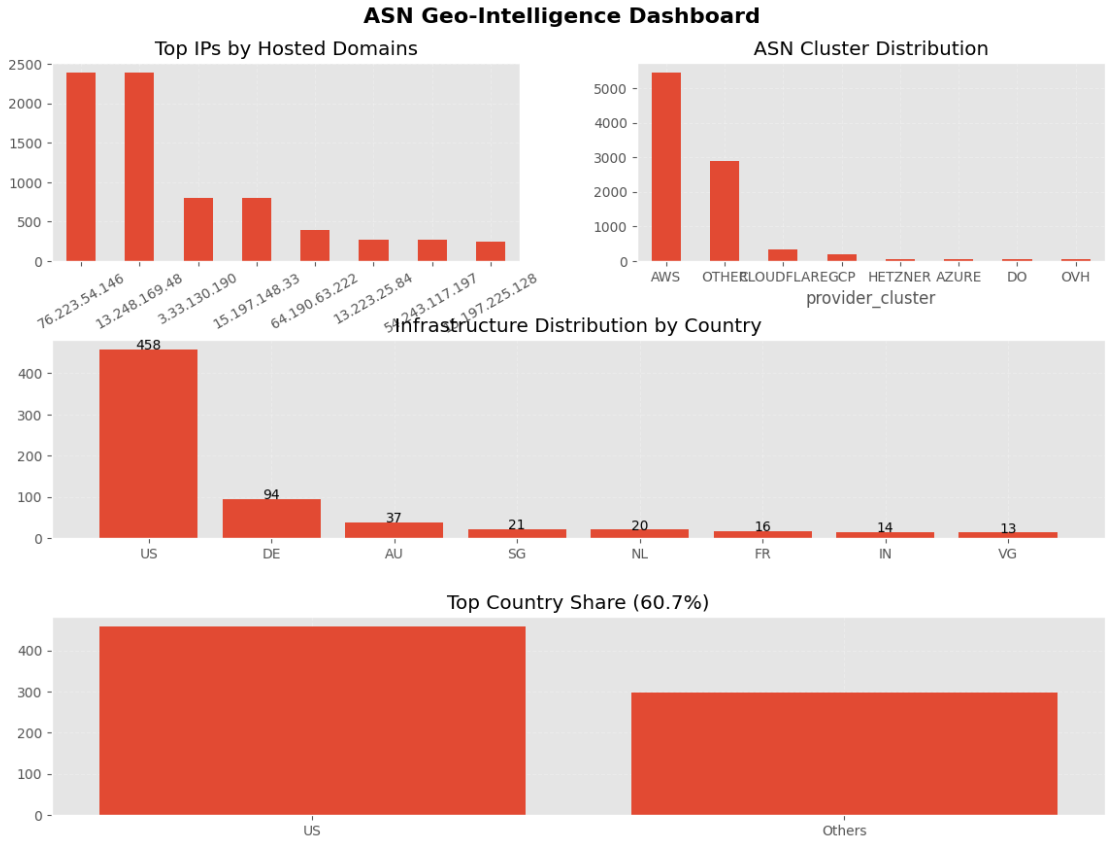

Figure 4.4.1: ASN Distribution — clustering of resolved domains across major cloud and infrastructure providers

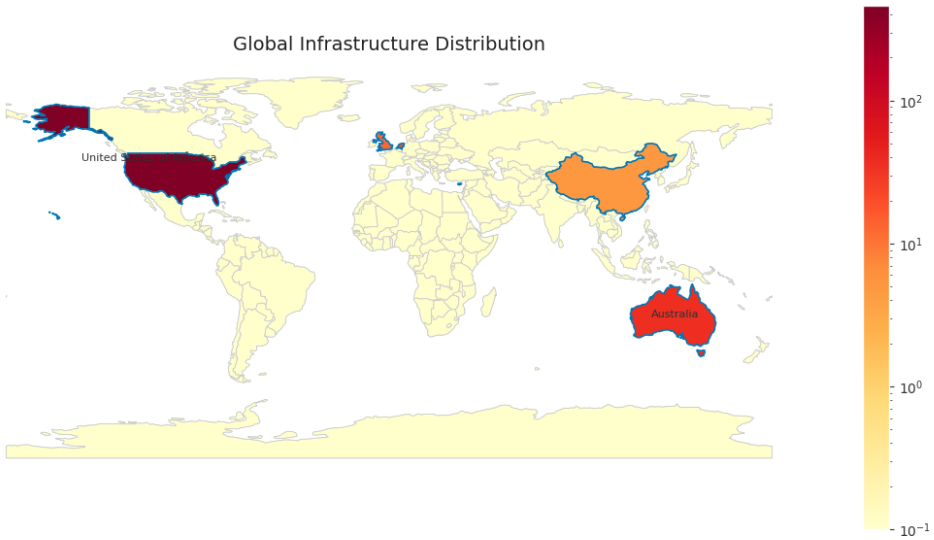

Figure 4.4.2: Global Infrastructure Distribution — geographic concentration of resolved domain infrastructure

The analysis reveals a strong concentration of resolved domains within a small number of major cloud and infrastructure providers. **AWS emerges as the dominant ASN cluster**, hosting approximately **5,400 domains**. This is followed by a generalized **OTHER** category, representing smaller providers, at around **3,000 domains**, and **Cloudflare** at approximately **1,200 domains**. Additional providers including **Google Cloud Platform (GCP)**, **Microsoft Azure**, **OVH**, **Hetzner**, and **DigitalOcean** account for the remaining share.

The geographic distribution of infrastructure, shown in Fig:4.4.2, highlights a clear concentration in the **United States**, with secondary presence in regions such as **Europe**, **Australia**, and parts of **Asia**. This aligns with the global footprint of major cloud providers and reinforces the observation that domain infrastructure is heavily centralized despite globally distributed domain targeting.

The concentration of domains within a limited set of large cloud providers is expected for active infrastructure, but it is also operationally significant. Providers such as AWS, Cloudflare, and GCP offer structured abuse reporting mechanisms that support coordinated mitigation when clusters of suspicious or malicious domains are identified. When clusters of lookalike or potentially malicious domains are identified within the same ASN or IP prefix, coordinated reporting can lead to rapid mitigation at the infrastructure level.

At the IP level, clustering further highlights infrastructure reuse. The top individual IP addresses by hosted domain count include **76.223.54.146** and **13.248.169.48**, each associated with approximately **2,400 domains**. Several additional IPs fall within the **750–1,000 domain** range.

Such high domain-per-IP ratios are characteristic of **shared hosting environments** or **domain parking infrastructure**, rather than individually managed deployments. This reinforces earlier observations from DNS provider analysis, where a significant portion of the resolved domain space is either parked or hosted on multi-tenant platforms.

From an intelligence perspective, ASN and IP clustering provide a powerful signal for identifying related infrastructure. Domains that appear unrelated at the lexical level may still share underlying hosting patterns, enabling correlation and grouping into actionable clusters. This capability is particularly useful for tracking campaign-level activity, where multiple domains are provisioned across the same infrastructure provider or IP range within a short time window.

4.5 System Performance

The performance characteristics of the system were monitored continuously throughout the run, capturing CPU utilisation, memory usage, query throughput, and task completion metrics. These metrics provide insight into system stability, resource efficiency, and the effectiveness of runtime controls such as rate limiting and cooldown mechanisms.

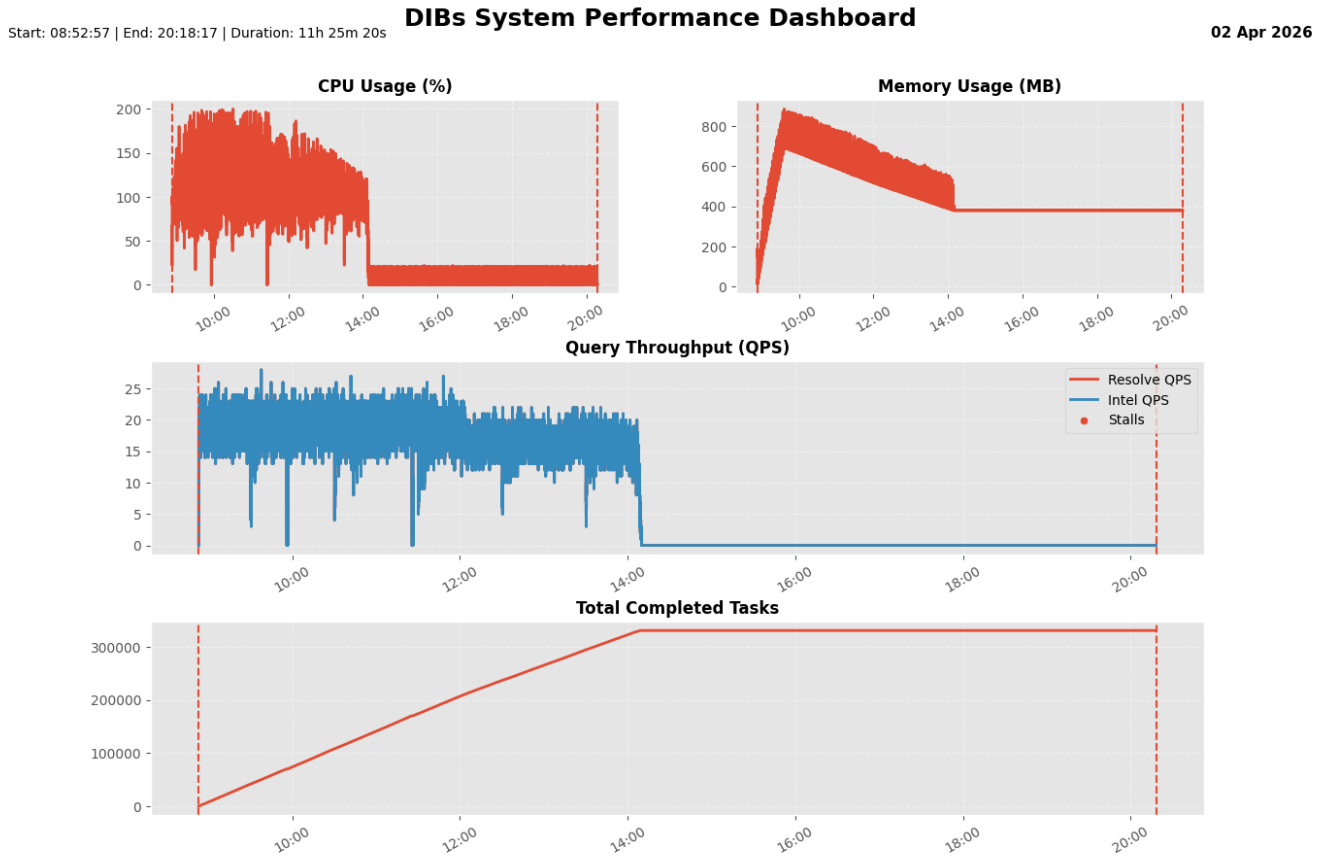

Figure 4.5.1: System Performance Metrics — CPU utilisation, memory usage, DNS query throughput (QPS), and task completion over time

CPU utilisation peaked at approximately **200%**, reflecting effective multi-core usage during the initial phase of the run when both domain generation and DNS resolution were active concurrently. As the generation spool was exhausted and the system transitioned into a resolution-dominant phase, CPU usage followed a gradual downward trend. The decline observed from approximately **12:00 onwards** indicates that the adaptive rate limiter correctly reduced processing intensity as the remaining workload decreased.

Memory usage peaked at approximately **800 MB** during the initial generation and resolution phase, then stabilised at around **400 MB** for the remainder of the run. This behaviour is consistent with the accumulation of in-memory structures, particularly the ASN cluster index containing approximately **12,000 resolved domain entries**, along with buffered writes from multiple concurrent NDJSON output streams. Importantly, the memory profile stabilised cleanly — no observable growth trend after the initial peak subsided, indicating the absence of memory leaks or uncontrolled allocation.

Query throughput was sustained at approximately **15–25 DNS queries per second** for both resolution and intelligence pipelines. The performance graph shows several brief stalls, highlighted as distinct interruptions in throughput. These correspond to **cooldown gate activations**, where the system intentionally paused execution for the configured **60-second** interval before resuming normal operation. This behaviour is expected and confirms that rate control mechanisms are functioning as designed rather than indicating instability.

Task completion followed a near-linear progression throughout the run, reaching approximately **330,000 completed tasks**. This aligns with the total generated domain count of **330,576**, accounting for entries already present in cache from prior runs or skipped due to spool reuse logic. The linear growth pattern further validates the stability of the worker pool and task scheduling system under sustained load.

Overall, the system demonstrates predictable and stable performance under high-throughput conditions. Resource utilisation remains controlled, adaptive mechanisms respond appropriately to workload changes, and no anomalous behaviour was observed during the execution window.

4.6 Execution Window and Idle Phase Behaviour

Although the total runtime extended from 08:52:57 to 20:18:17 IST, all active processing — including domain generation, DNS resolution, and intelligence enrichment — was completed by approximately 14:09 IST.

This behaviour is consistent with the linear task completion pattern observed in Fig 4.5.1. Following this point, the system remained operational but idle, with no pending tasks in the generation, resolution, or intelligence queues. This is reflected in the performance metrics, where CPU utilisation and query throughput drop to near-zero levels for the remainder of the runtime.

The extended runtime beyond the active processing window does not indicate performance degradation or stalled execution. Instead, it reflects the system remaining in a stable idle state after successful completion of all scheduled tasks.

5 Limitations and Known Gaps

No system description is complete without a clear account of its limitations. The following gaps are known and observed during implementation and testing. Some represent deliberate design trade-offs accepted in the interest of simplicity or scope control, while others are areas identified for future improvement. Nine limitations are documented below, covering runtime behaviour, infrastructure dependencies, and output management.

5.1 Adaptive Control is Incomplete

The adaptive controller in `core/adaptive/` is fully implemented and unit-tested. It computes EWMA-based pressure scores derived from queue depth, in-flight task counts, and latency samples, and produces recommendations for rate limits and timeout adjustments.

However, the `observeTask` and `observeLimiterError` hooks in the runtime tuner are currently unimplemented stubs. As a result, the feedback loop remains architecturally complete but functionally disconnected. Enabling `autoscale=true` in the configuration has no effect in the current build. This represents the highest-priority gap for the next development cycle.

5.2 SQLite Spool Persistence Instability

The SQLite spool mechanism is currently disabled in the production build due to non-deterministic behaviour observed under sustained load conditions during testing. The issue is isolated to the persistence layer and

does not impact the correctness of domain generation or downstream resolution. Re-enablement is planned following validation of deterministic write guarantees.

5.3 Intel Pipeline Parallelism is Limited

Within the intelligence pipeline, each domain is processed using six parallel goroutines to fetch DNS record types (`A`, `AAAA`, `CNAME`, `MX`, `NS`, `TXT`) concurrently. However, the pipeline itself processes only one domain at a time from the Redis `BLPOP` queue.

Under high-throughput conditions, this creates a bottleneck where resolved domains accumulate faster than they can be processed. The correct solution is to introduce a worker pool within the intel pipeline, allowing multiple domains to be processed in parallel. This limitation was intentionally accepted in the initial implementation to maintain simplicity but becomes significant at scale.

5.4 No Cross-Run Deduplication

The in-memory cache (`resolvedSeen sync.Map`) prevents duplicate intelligence processing within a single run. However, it does not persist across executions.

If the same spool is reused after a restart, domains that were previously enriched will be processed again by the intel pipeline. While DNS resolution is correctly cached in Redis (with a 48-hour TTL for successful resolutions and 10 seconds for failures), the enrichment phase is repeated, leading to duplicate records in output files.

A persistent seen-set, stored in Redis or SQLite, would eliminate this redundancy.

5.5 Redis is a Hard Dependency

Redis is required for three core functions: rate limiting, DNS resolution caching, and the intel processing queue. If Redis is unavailable at startup, the system attempts to install and launch it, but no fallback mechanism exists.

A Redis failure during execution will stall both the rate limiter and the intel pipeline. For environments where Redis availability cannot be guaranteed, this introduces operational risk.

A lightweight in-process fallback, such as an atomic token bucket for rate limiting and an in-memory queue for intel processing, would remove this dependency.

5.6 NDJSON Output Has No Rotation or Size Limits

All output files grow continuously for the duration of a run. There is no file rotation, size capping, or compression mechanism.

While this has not posed issues in current test scenarios, large-scale or continuous runs will produce significantly large files. This behaviour is consistent with the design choice of avoiding opinionated storage layers, but it requires operators to manage disk usage externally.

5.7 DNSSEC Validation is Not Verified

The DNSSEC field in intelligence records reflects resolver-reported status rather than verified chain-of-trust validation. The system does not perform full DNSSEC validation from the root.

This is sufficient for large-scale presence detection but should not be interpreted as definitive cryptographic validation. Operators requiring validated DNSSEC guarantees must perform independent verification.

5.8 Timestamps are Fixed to IST

All timestamps in NDJSON outputs are recorded in IST (UTC+5:30). This is currently hardcoded and not configurable.

Operators in different timezones, or systems expecting UTC, must normalise timestamps during data ingestion. While trivial to correct in implementation, this has not yet been addressed.

5.9 API Has No TLS and is Localhost-Scoped

The API server binds to `127.0.0.1:9090` and does not provide TLS. Authentication is handled via ed25519 key validation, but communication remains unencrypted.

This configuration is intentional for local-only operation. Any remote access scenario requires external mechanisms such as SSH tunnelling or reverse proxies to provide transport security. Direct exposure of the API without TLS is not supported.

6 Reproducibility

All results presented in this paper are derived from a single execution run conducted on 2 April 2026 using the DIBs framework under controlled conditions.

To ensure both reproducibility and independent verification, the complete execution artifact has been preserved and publicly released. This artifact includes the exact binary, input datasets, configuration files, execution logs, and output data generated during the original run.

Source Code: `https://github.com/Mr-Biswadeb-Mukherjee/DIBs` [5]

Release Artifact: `https://github.com/Mr-Biswadeb-Mukherjee/DIBs`

The release package contains:

- Compiled binary used during the experiment
- Exact keyword input dataset (`Keywords.csv`)
- Configuration files including runtime and DNS settings
- Full execution logs covering application, DNS resolution, and rate limiting
- Generated outputs including NDJSON intelligence records, resolved domain data, and spool database files

This enables two modes of validation. First, the original execution can be directly verified using the preserved logs and outputs. Second, the experiment can be re-executed using the provided source code and configuration under equivalent system and network conditions.

Sample datasets are included within the release package. Full datasets can be regenerated using the provided inputs and configuration.

This approach provides both *reproducibility* through complete system availability and *verifiability* through preservation of the original execution artifact.

7 Conclusion

DIBs achieves its intended objective: generating and enriching a comprehensive domain intelligence dataset from a moderate keyword set within a single day on commodity hardware, without reliance on external services, cloud infrastructure, or proprietary tooling.

From a total of **330,576 generated domain candidates**, the system produced **12,860 resolved domains**, yielding a **3.9% resolution rate**. Each resolved domain was enriched with multi-record DNS data, ASN attribution, provider classification, and WHOIS metadata. The resulting dataset forms a structured and immediately usable intelligence layer, with NDJSON outputs designed for direct ingestion into downstream analysis pipelines.

Several observations from the results are particularly relevant:

- **High concentration of parked domains.** Afternic and `domaincontrol.com` together account for approximately **31.8%** of all resolved domains. This indicates that a significant portion of the mutation space is already registered and held in a parked state. Although inactive, these domains represent pre-positioned infrastructure that can be activated rapidly with minimal changes.
- **Zero DNSSEC adoption.** None of the resolved domains had DNSSEC enabled. While this is consistent with the nature of lookalike and parked domains, it establishes a baseline for future comparative analysis and highlights the absence of integrity protections across the dataset.
- **Concentration in major cloud providers.** Infrastructure is heavily clustered within providers such as AWS, Cloudflare, and GCP. This is expected, but operationally important, as these providers expose abuse reporting channels that can be leveraged when clusters of malicious or suspicious domains are identified.
- **Stable and predictable performance.** CPU peaked at $\sim$200% during generation, memory stabilised at $\sim$400 MB, post-peak, and cooldown mechanisms triggered as designed throughout the run.

The areas for extension are clearly defined. The adaptive controller is implemented and tested, but not yet integrated into the runtime tuning loop. Enabling autoscaling would allow the system to dynamically adjust throughput based on available system capacity. The intelligence pipeline, while parallelised at the record level, processes domains sequentially; introducing domain-level parallelism would reduce latency and improve throughput under high load.

Additionally, introducing a structured diff capability between runs would enable delta detection without requiring full reprocessing, further enhancing operational efficiency.

The codebase is supported by unit and integration tests covering mutation algorithms, DNS resolution stubs, spool resumption logic, API contract validation, rate limiter behaviour, and ASN cluster indexing. A continuous integration pipeline validates the full build before changes are merged, ensuring baseline stability.

DIBs is designed as research infrastructure rather than a managed product. It prioritises transparency, extensibility, and control. The system is intended for engineers who need to generate, inspect, and understand domain intelligence at scale, and who are prepared to adapt and extend the tooling to fit their specific operational requirements.

8 About the Author

Biswadeb Mukherjee is an Offensive Security Specialist and Malware Engineer with over five years of experience in adversary simulation, offensive tooling research and development, OSINT operations, and dark web threat intelligence. All research and technical experimentation are conducted within controlled and isolated lab environments.

9 License

How to Cite

Mukherjee, Biswadeb. *Domain Intelligence & Behaviour System (DIBs)*. Technical Whitepaper, Version Version 1.0.1, April 6, 2026.
Available at: https://official-biswadeb941.in

BibTeX:

@techreportoffsec-biswadeb2026-dibsv1, author = Mukherjee, Biswadeb, title = Domain Intelligence & Behaviour System (DIBs), year = 2026, version = 1.0, url = https://official-biswadeb941.in, note = Technical Whitepaper

References

[1] Verisign, Inc. Domain name industry brief. https://www.verisign.com/en_US/domain-names/dnib/index.xhtml, 2025.

[2] Paul Mockapetris. Domain names - concepts and facilities. https://datatracker.ietf.org/doc/html/rfc1034, 1987.

[3] Paul Mockapetris. Domain names - implementation and specification. https://datatracker.ietf.org/doc/html/rfc1035, 1987.

[4] ICANN. Domain name system (dns) overview. https://www.icann.org/resources/pages/dns-2012-02-25-en, 2024.

[5] Mukherjee, Biswadeb. Domain Intelligence & Behaviour System (DIBs). https://github.com/Mr-Biswadeb-Mukherjee/DIBs. GitHub repository, 2026.

[6] Matthew A. Jaro. Advances in record-linkage methodology. *Journal of the American Statistical Association*, 84(406):414–420, 1989.

[7] William E. Winkler. String comparator metrics and enhanced decision rules in the fellegi-sunter model of record linkage. *Proceedings of the Section on Survey Research Methods*, pages 354–359, 1990.

[8] Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3):379–423, 1948.

[9] Miek Gieben. miekg/dns: Dns library for go. https://github.com/miekg/dns, 2024.

[10] ICANN. Whois and rdap overview. https://www.icann.org/resources/pages/whois-rdap-2018-04-27-en, 2024.

[11] R. Arends et al. Dns security introduction and requirements. https://datatracker.ietf.org/doc/html/rfc4033, 2005.

[12] R. Arends et al. Resource records for dns security extensions. https://datatracker.ietf.org/doc/html/rfc4034, 2005.

[13] R. Arends et al. Protocol modifications for dns security extensions. https://datatracker.ietf.org/doc/html/rfc4035, 2005.

[14] Cloudflare, Inc. What is dnssec? https://www.cloudflare.com/learning/dns/dnssec/, 2024.