

Declarative Rule Policy Engine

STATE**published****CREATED**

2026-09-07

LAST UPDATED

2026-09-07

INTENDED STATUS

Published

AUTHORS

Sibuna Contributors <team@sibuna.local>

DISCUSSION

Specification of Sibuna's zero-allocation declarative rule engine: JSON policy files, multi-criteria matching over path, user agent, headers, and IPv4/IPv6 CIDRs, WEIGH scoring with thresholds, per-rule challenge parameters, the evaluation order, and dynamic policies replicated through Zaxonlite.

LABELS

policy, architecture, zero-alloc, firewall

Status of This Memo

This document is an internal Sibuna discussion record authored in Typst and tracked in git. It intentionally follows RFC-style structure so the design scope, rationale, trade-offs, and operational constraints remain explicit. Documents that still use the placeholder number XXXXX are provisional drafts. Numbered SID documents are part of the permanent project record.

Contents

Abstract	1
Introduction	1
The SID Lifecycle	1
Tooling and Numbering Workflow	1
Creating a Draft	1
Promoting a Draft	1
Listing Discussions	1
Engineering Principles and Code Standards	1
Hard Structural Limits	1
Core TigerStyle Priorities	1
Elm-Style Diagnostic and Error Reporting	1
The Three Elements of an Elm-Style Error	1
Concrete Diagnostic Example	1

Implementation Architecture	1
Recoverable Errors vs Invariant Assertions	1
Compilation Targets and Verification	1
Conclusion	1
Decision	1
Request contract	1
Authentication and work	1
Policy and memory	1
Measurement gates	1
Release toolchain	1
Native platforms and release boundaries	1
Remaining limits	1
1. Abstract	9
2. Introduction	9
3. The SID Lifecycle	9
4. Tooling and Numbering Workflow	10
4.1. Creating a Draft	10
4.2. Promoting a Draft	10
4.3. Listing Discussions	10
5. Engineering Principles and Code Standards	10
5.1. Hard Structural Limits	10
5.2. Core TigerStyle Priorities	11
6. Elm-Style Diagnostic and Error Reporting	11
6.1. The Three Elements of an Elm-Style Error	11
6.2. Concrete Diagnostic Example	11
6.3. Implementation Architecture	12
6.4. Recoverable Errors vs Invariant Assertions	12
7. Compilation Targets and Verification	12
8. Conclusion	12
9. Decision	9
10. Request contract	10
11. Authentication and work	11
12. Policy and memory	11
13. Measurement gates	11
14. Release toolchain	12
15. Native platforms and release boundaries	12
16. Remaining limits	12
Decision summary	1
Data model	1
Evaluation order	1
Built-in policy	1
Policy file	1
Dynamic policies	1
Verification	1
Milestones	1
Review corrections (2026-09-07)	1
17. Decision summary	9
18. Data model	9
19. Evaluation order	10

20. Built-in policy	10
21. Policy file	11
22. Dynamic policies	12
23. Verification	12
24. Milestones	12
25. Review corrections (2026-09-07)	12
Context and problem statement	1
Engine choice: automata and tokenizers, not regular expressions	1
Design	1
Byte-class pre-scan	1
26. Context and problem statement	9
27. Engine choice: automata and tokenizers, not regular expressions	9
28. Design	10
28.1. Byte-class pre-scan	10
Context and motivation	1
Zaxonlite foundations	1
Build and configuration	1
Schema	1
Hot-path isolation	1
Read-copy-update engine slot	1
Incident ring	1
Storage thread tick	1
Campaign clustering	1
Forensics	1
Product boundary	1
Delivery record	1
Verification	1
Open items	1
Review corrections (2026-09-07)	1
29. Context and motivation	9
30. Zaxonlite foundations	10
31. Build and configuration	10
32. Schema	10
33. Hot-path isolation	11
33.1. Read-copy-update engine slot	11
33.2. Incident ring	11
33.3. Storage thread tick	11
34. Campaign clustering	12
35. Forensics	12
36. Product boundary	12
37. Delivery record	12
38. Verification	12
39. Open items	13
40. Review corrections (2026-09-07)	13
Abstract	1
Introduction and Adversary Model	1
Setting	1
Axioms	1
Cost calculus	1

Part A: The Hashcash Tier	1
Part B: Proof of Sequential Work	1
Why sequential work	1
The construction	1
Theorems	1
Why not the alternatives	1
Implementation and calibration	1
Part C: Stateless Challenges and Symmetric Tokens	1
Pseudorandom functions and message authentication	1
Stateless challenge identifiers	1
Session tokens	1
Part D: Rate Limiting by the Generic Cell Rate Algorithm	1
Part E: Robin Hood Hashing for the Spent Set	1
Part F: Linear-Time Inspection	1
Multi-pattern matching	1
The semantic layer	1
Vectorised matchers	1
Part G: Read-Copy-Update Engine Slots	1
Part H: Load-Adaptive Difficulty	1
Part I: Campaign Clustering by Feature Hashing	1
Verification and Benchmark Gates	1
Security Considerations	1
Open Problems	1
References	1
41. Abstract	9
42. Introduction and Adversary Model	10
42.1. Setting	10
42.2. Axioms	10
42.3. Cost calculus	11
43. Part A: The Hashcash Tier	11
44. Part B: Proof of Sequential Work	12
44.1. Why sequential work	12
44.2. The construction	12
44.3. Theorems	13
44.4. Why not the alternatives	14
44.5. Implementation and calibration	15
45. Part C: Stateless Challenges and Symmetric Tokens	15
45.1. Pseudorandom functions and message authentication	15
45.2. Stateless challenge identifiers	16
45.3. Session tokens	17
46. Part D: Rate Limiting by the Generic Cell Rate Algorithm	17
47. Part E: Robin Hood Hashing for the Spent Set	18
48. Part F: Linear-Time Inspection	19
48.1. Multi-pattern matching	19
48.2. The semantic layer	20
48.3. Vectorised matchers	20
49. Part G: Read-Copy-Update Engine Slots	21
50. Part H: Load-Adaptive Difficulty	21
51. Part I: Campaign Clustering by Feature Hashing	22

52. Verification and Benchmark Gates	22
53. Security Considerations	23
54. Open Problems	23
55. References	24
Abstract	1
Introduction and Motivation	1
Operator Workflows	1
Design Principles	1
Don't make me think	1
Fast judgement: System 1	1
Careful judgement: System 2	1
Elegance	1
Principle audit by page	1
Terminology and Scope	1
Goals and Non-Goals	1
Goals	1
Non-Goals	1
Architecture Overview	1
Modules and directories	1
Process model and the isolation contract	1
Startup from the command line	1
The Serve Kernel	1
Authentication and Authorization	1
The Telemetry Pipeline	1
The sampler	1
The incident tap	1
The challenge funnel	1
Traffic sampling and top-k rankings	1
Data-plane changes this record requests	1
Cluster aggregation	1
The Real-Time Protocol	1
Data Model	1
GeoIP	1
Live earth globe on the landing page	1
Cluster Management	1
The Interface Module	1
Pages	1
Wireframes	1
The Build Pipeline	1
Security Considerations	1
Performance Budget	1
Delivery Plan	1
Verification	1
Resolved Design Decisions	1
Operational Contracts and Trade-offs	1
Retained observations	1
Rule observations	1
Management consistency	1
Browser and peer state	1

Acceptance and Open Questions	1
Query and authentication observation bounds	1
Launch scope (2026-09-11)	1
Post-launch completion (2026-09-11)	1
Interface budget review (2026-09-11)	1
References	1
56. Abstract	9
57. Introduction and Motivation	10
58. Operator Workflows	10
59. Design Principles	11
59.1. Don't make me think	11
59.2. Fast judgement: System 1	12
59.3. Careful judgement: System 2	13
59.4. Elegance	13
59.5. Principle audit by page	13
60. Terminology and Scope	14
61. Goals and Non-Goals	15
61.1. Goals	15
61.2. Non-Goals	15
62. Architecture Overview	16
62.1. Modules and directories	16
62.2. Process model and the isolation contract	17
62.3. Startup from the command line	18
63. The Serve Kernel	19
64. Authentication and Authorization	20
65. The Telemetry Pipeline	21
65.1. The sampler	21
65.2. The incident tap	22
65.3. The challenge funnel	23
65.4. Traffic sampling and top-k rankings	23
65.5. Data-plane changes this record requests	24
65.6. Cluster aggregation	24
66. The Real-Time Protocol	24
67. Data Model	26
68. GeoIP	28
68.1. Live earth globe on the landing page	29
69. Cluster Management	30
70. The Interface Module	30
70.1. Pages	31
71. Wireframes	33
72. The Build Pipeline	43
73. Security Considerations	44
74. Performance Budget	45
75. Delivery Plan	46
76. Verification	47
77. Resolved Design Decisions	48
78. Operational Contracts and Trade-offs	49
78.1. Retained observations	49
78.2. Rule observations	49

78.3. Management consistency	50
78.4. Browser and peer state	50
79. Acceptance and Open Questions	51
79.1. Query and authentication observation bounds	51
79.2. Launch scope (2026-09-11)	51
79.3. Post-launch completion (2026-09-11)	52
79.4. Interface budget review (2026-09-11)	53
80. References	53
Abstract	1
Introduction and Motivation	1
Terminology and Scope	1
Problem Statement and Architectural Challenges	1
The Gap in the Current Codebase	1
The Asymmetric Challenges of AI Bot Identification	1
System Invariants and Design Principles	1
Detailed Design	1
1. The Multi-Tier Bot Classification Engine	1
1.1 Provider and Bot Taxonomy	1
1.2 Comprehensive AI Bot Signatures Database	1
1.3 Sub-Microsecond Radix CIDR Verification	1
1.4 Behavioral and Structural Heuristics	1
2. High-Efficiency Streaming Analytics: Algorithms, Mathematics, and Storage Decoupling	1
2.1 The Analytics Dilemma: Why Traditional OLAP Fails Edge WAFs	1
2.2 Axiomatic Foundations of Edge Analytics	1
2.3 Contention-Free Cache-Line Striped Accumulation	1
2.4 Bounded Stochastic Sampling and Poisson Ring Dynamics	1
2.5 Space-Saving Heavy-Hitters Algorithm for Path Scraping	1
2.6 Micro-RRD Pre-Aggregated Rollups and Zaxonlite Storage Decoupling	1
2.7 Long-Term Telemetry Egress Strategy	1
3. Operator Console Dashboard Interface	1
3.1 Operator Screen Wireframes	1
3.2 Visual Components	1
3.3 Granular Tabular Components	1
3.4 Actionable Policy Integration	1
4. Elm-Style Operator Error and Diagnostic Reporting	1
Verification and Testing Matrix	1
Security Considerations	1
Rollout and Operational Plan	1
References	1
81. Abstract	9
82. Introduction and Motivation	9
83. Terminology and Scope	10
84. Problem Statement and Architectural Challenges	11
84.1. The Gap in the Current Codebase	11
84.2. The Asymmetric Challenges of AI Bot Identification	12
85. System Invariants and Design Principles	12
86. Detailed Design	13
86.1. 1. The Multi-Tier Bot Classification Engine	13

86.1.1.	1.1 Provider and Bot Taxonomy	13
86.1.2.	1.2 Comprehensive AI Bot Signatures Database	14
86.1.3.	1.3 Sub-Microsecond Radix CIDR Verification	15
86.1.4.	1.4 Behavioral and Structural Heuristics	16
86.2.	2. High-Efficiency Streaming Analytics: Algorithms, Mathematics, and Storage	
	Decoupling	16
86.2.1.	2.1 The Analytics Dilemma: Why Traditional OLAP Fails Edge WAFs	16
86.2.2.	2.2 Axiomatic Foundations of Edge Analytics	17
86.2.3.	2.3 Contention-Free Cache-Line Striped Accumulation	17
86.2.4.	2.4 Bounded Stochastic Sampling and Poisson Ring Dynamics	18
86.2.5.	2.5 Space-Saving Heavy-Hitters Algorithm for Path Scraping	19
86.2.6.	2.6 Micro-RRD Pre-Aggregated Rollups and Zaxonlite Storage Decoupling . . .	20
86.2.7.	2.7 Long-Term Telemetry Egress Strategy	21
86.3.	3. Operator Console Dashboard Interface	22
86.3.1.	3.1 Operator Screen Wireframes	22
86.3.2.	3.2 Visual Components	24
86.3.3.	3.3 Granular Tabular Components	25
86.3.4.	3.4 Actionable Policy Integration	25
86.4.	4. Elm-Style Operator Error and Diagnostic Reporting	26
87.	Verification and Testing Matrix	26
88.	Security Considerations	26
89.	Rollout and Operational Plan	27
90.	References	27
	Abstract	1
	Introduction	1
	Terminology and Scope	1
	Problem Statement	1
	Goals and Non-Goals	1
	Goals	1
	Non-Goals	1
	State of the Art	1
	Protocol Rules	1
	Detailed Design	1
	The decoder	1
	Admission: decoding into the head buffer	1
	Forwarding	1
	Cost	1
	Verification and Testing	1
	Security Considerations	1
	Rollout and Operational Plan	1
	Implementation Record (2026-10-02)	1
	References	1
91.	Abstract	9
92.	Introduction	9
93.	Terminology and Scope	10
94.	Problem Statement	10
95.	Goals and Non-Goals	10
	95.1. Goals	10
	95.2. Non-Goals	11

96. State of the Art	11
97. Protocol Rules	11
98. Detailed Design	12
98.1. The decoder	12
98.2. Admission: decoding into the head buffer	13
98.3. Forwarding	14
98.4. Cost	15
99. Verification and Testing	15
100. Security Considerations	15
101. Rollout and Operational Plan	16
102. Implementation Record (2026-10-02)	16
103. References	17

Revision note (2026-09-07)

Revised after the implementation review of 2026-09-07. The original default rule table admitted any User-Agent containing Mozilla and fell through to ALLOW; that contradicted the product principle that an unverified client must pay for admission, and it meant a scraper with a browser User-Agent was never challenged. The implemented defaults, WEIGH semantics, JSON schema, IPv6 support, and evaluation order are recorded here as built.

17. Decision summary

Sibuna evaluates every request against an ordered table of declarative rules compiled at startup (or rebuilt off the hot path from the Zaxonlite policies table, SID 0005). A rule is a conjunction of optional criteria over the request path, the User-Agent, up to four headers, and up to eight IPv4 or IPv6 CIDR blocks. Its action is ALLOW, DENY, CHALLENGE, or WEIGH, with optional per-rule challenge difficulty (work bits) and algorithm (hashcash or posw). Evaluation is first-terminal-match-wins; WEIGH rules contribute a signed score that is resolved against thresholds when no terminal rule matched. The engine holds at most 128 rules and never allocates during evaluation.

Performance contract

Evaluation uses fixed-capacity tables and borrowed request slices with no allocator argument. Exact engine memory and current Gate/Shield timings are recorded by the standalone harness in benchmarks/results/latest.json. These primitive timings exclude socket I/O and do not establish HTTP throughput or instrumented allocation counts.

18. Data model

```
pub const Action = enum(u8) { allow, deny, challenge, weigh };

pub const HeaderMatcher = struct { name: []const u8, pattern: []const u8 };
pub const CidrMatcher = struct { network: u128, mask: u128 }; // IPv4 mapped
into ::ffff:0:0/96

pub const PolicyRule = struct {
    name: []const u8,
    path_pattern: ?[]const u8 = null,
    ua_pattern: ?[]const u8 = null,
```

```

headers: [4]HeaderMatcher, header_count: u8 = 0,
cidrs: [8]CidrMatcher, cidr_count: u8 = 0,
action: Action = .allow,
difficulty: ?u32 = null,          // work bits
algorithm: ?[]const u8 = null,    // "hashcash" | "posw"
weight: i32 = 0,                 // WEIGH contribution, may be negative
};

pub const RequestView = struct {
    path: []const u8, query: []const u8 = "", client_ip: []const u8,
    user_agent: []const u8 = "", headers: []const Header = &.{}, body: []const u8
= "",
};

pub const Decision = struct {
    action: Action, rule_name: []const u8, difficulty: u32,
    algorithm: ?[]const u8 = null, score: i32 = 0,
};

```

Pattern grammar: .* or * matches anything; ^...\$ anchors an exact path; a trailing *, /*, or .* is a prefix match; a pattern beginning with / is an exact path; anything else is a case-insensitive substring, the natural form for User-Agent rules. Header patterns use the same grammar against the header value. CIDR matchers accept 10.0.0.0/8, 2001:db8::/32, bare addresses, and IPv4-mapped IPv6 literals.

19. Evaluation order

Engine.evaluateRequest(RequestView) Decision runs:

1. **Semantic WAF** (Shield surface only, SID 0004): any violation is a terminal DENY named waf:<category>.
2. **Reputation trie, terminal verdicts**: an IP matching a deny or allow prefix returns immediately as ip/cidr-trie. Bans propagated from storage therefore beat every rule, including the generic-browser challenge.
3. **Declarative rules in order**: the first ALLOW/DENY/CHALLENGE match returns; WEIGH matches add weight to the score and continue.
4. **Score resolution** (when the score is non-zero): negative totals ALLOW; totals at or above deny_at (default 40) DENY; totals at or above challenge_at (default 10) CHALLENGE with default_difficulty + min((score - challenge_at) / bits_step, max_extra_bits) work bits (bits_step 5, max_extra_bits 6).
5. **Static bypass paths**: /favicon.ico, /robots.txt, /.well-known/*, /__sibuna/*.
6. **Reputation trie, challenge verdict**.
7. **Bot-signature automaton**: a match is CHALLENGE named after the signature.
8. **Default action**: CHALLENGE unless the policy file sets default_action.

20. Built-in policy

Rule	Criteria	Action	Purpose
well-known	Path ^/.well-known/.*\$	ALLOW	ACME and security.txt
favicon	Path ^/favicon.ico\$	ALLOW	Browser icon fetch
robots-txt	Path ^/robots.txt\$	ALLOW	Robots exclusion standard

Rule	Criteria	Action	Purpose
sibuna-internal	Path /__sibuna/*	ALLOW	Solver assets and APIs
Serverless-header denial	Header CF-Worker: .*	DENY	Unconsented serverless scrapers
amazonbot	UA contains Amazonbot	DENY	Aggressive harvester
generic-browser	UA contains Mozilla	CHALLENGE	Humans clear the interstitial; scrapers pay
(bot automaton)	UA matches an AI-scraper or scraper-library signature	CHALLENGE	Known automation
(default)	Anything else	CHALLENGE	Unknown clients must prove work

Why browsers are challenged

A browser User-Agent is free to forge. If Mozilla admitted a client, every scraper would send it and the proof-of-work gate would be decorative. The interstitial costs a human 15–150 ms once per session; a harvesting fleet pays it for every session it opens. Operators who need unauthenticated API traffic admit it explicitly by path, header, or CIDR rule, or set "default_action": "ALLOW" to run the engine as a pure block list.

21. Policy file

--policy-file <path> (or -P) loads a JSON document at startup. An explicit rules array replaces the built-in rule table; the bypass paths, reputation trie, and bot automaton remain.

```
{
  "default_action": "CHALLENGE",
  "waf": true,
  "thresholds": { "challenge_at": 10, "deny_at": 40, "bits_step": 5 },
  "ip_rules": { "10.0.0.0/8": "ALLOW", "2001:db8::/32": "DENY" },
  "rules": [
    { "name": "deny-bad-worker", "headers": { "CF-Worker": ".*" }, "action":
"DENY" },
    { "name": "protect-checkout", "path": "/api/checkout/*", "action":
"CHALLENGE",
      "challenge": { "difficulty": 20, "algorithm": "posw" } },
    { "name": "internal-subnets", "remote_addresses": ["10.0.0.0/8", "fd00::/8"],
"action": "ALLOW" },
    { "name": "headless", "user_agent": "Headless", "action": "WEIGH", "weight":
30 },
    { "name": "partner-token", "headers": { "X-Partner": "v2" }, "action":
"WEIGH", "weight": -20 }
  ]
}
```

Field aliases accepted for compatibility with existing policy files: path_regex, user_agent_regex, headers_regex, cidrs. ip_rules feeds the reputation trie, which scales to thousands of prefixes; per-rule remote_addresses is meant for a handful.

22. Dynamic policies

With `--data-dir`, rows of the Zaxonlite policies table (name, priority, patterns, action, difficulty, algorithm, JSON header matchers, JSON CIDR list, weight, enabled) are appended after the file rules on every rebuild, and rebuilds are published to the workers through the read-copy-update engine slot without a restart. Rows replicate across a cluster by Multi-Paxos. The mechanism is specified in SID 0005.

23. Verification

- Unit tests cover pattern grammar, IPv4 and IPv6 CIDR matching, rule conjunction, WEIGH accumulation into allow, challenge with extra bits, and deny, JSON loading including waf, thresholds, ip_rules, and weight, and a 100,000-request zero-allocation loop.
- End-to-end tests exercise the built-in denials (crawler and serverless-header denials), the static bypass on `/robots.txt` with a bot User-Agent, and the interstitial for a browser.
- The storage test inserts a policy row with header and CIDR matchers and observes the rebuilt engine apply it with the row's difficulty and algorithm.

24. Milestones

M1: Rule data structures and zero-allocation matchers delivered

Outcome: ``PolicyRule``, ``Action``, path/header/CIDR matching over stack slices, IPv6 support.

Exit criterion: Unit tests pass with zero heap allocation.

M2: JSON loader and default table delivered

Outcome: ``Engine.loadFromJsonInto``, thresholds, ``ip_rules``, WEIGH weights, default challenge posture.

Exit criterion: Loader test and engine tests pass.

M3: Server and coordinator integration delivered

Outcome: ``--policy-file``, rule difficulty and algorithm carried in the challenge id, rule hash bound into the session token.

Exit criterion: End-to-end tests observe per-rule challenge parameters.

M4: Dynamic policies delivered

Outcome: Zaxonlite ``policies`` table rebuilt into the spare engine and published by RCU.

Exit criterion: Storage test observes a database rule after one tick.

25. Review corrections (2026-09-07)

The daemon evaluates policy before accepting a session, so a session bypasses only the challenge verdict. Dynamic database rules are ordered before file/default rules to keep a generic browser challenge from masking an operator's denial. Exact anchored patterns match the complete string. Rule names must contain 1–128 bytes and no control bytes, bounding response copies and

preventing injected audit headers. The engine uses pattern-derived automaton capacity. Current timings are recorded in `benchmarks/results/latest.json`; older figures above describe the previous build.