Declarative Rule Policy Engine

STATE
published
TYPE
Architectural Specification
CREATED
2026-09-07
SUMMARY
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.

STATUS
Published
AUTHORS
Sibuna Contributors <team@sibuna.local>
UPDATED
2026-10-15

DISCUSSION
No external discussion URL assigned

LABELS
policy, architecture, zero-alloc, firewall

Record status

The catalog defines the review process and the meaning of each state. This numbered record forms part of the catalog.

Contents

1. Revision note 1
2. Abstract 2
3. Decision summary 3
4. Data model 3
5. Evaluation order 3
6. Built-in policy 4
7. Policy file 4
8. Dynamic policies 5
9. Verification 5
10. Milestones 5
11. Review corrections (2026-09-07) 6
12. References 6

1. Revision note

2026-10-15: Ported from the Typst original in the Zig repository. The record describes the Zig implementation as it was decided; SID 0010 maps its packages to the Rust crates of the port.

2026-10-15 (Rust port): the reputation trie comes from `sibuna-cidr`, which fixes a Zig bug. The Zig trie stored the prefix `0.0.0.0/0` (equally `::ffff:0:0/96`) on the node where IPv4 lookups begin, but never read that node's value, so an `ip_rules` entry for it was accepted and silently matched nothing. In the port it matches every IPv4 client, as its text says. A policy file that carries such an entry therefore behaves differently: `"0.0.0.0/0": "DENY"` now denies every IPv4 client at step 2 of the evaluation order, where the Zig daemon

ignored it. Per-rule `remote_addresses` were matched by mask arithmetic in Zig and were not affected. A failed trie insertion now leaves the trie unchanged; Zig kept the nodes it had created, and they counted against the 8,192-node capacity. The accepted address syntax, including Zig's lenient IPv4 parts such as `010` and `+1`, is unchanged.
2026-10-15 (Rust port): score resolution is computed in wider, saturating arithmetic, so extreme managed weights and difficulties saturate where Zig would have trapped on overflow; every value a valid policy produces is unchanged. A number in an integer field that is whole or not depending on binary128 rounding (more than about 34 significant digits) is judged exactly, where Zig could round it to an integer and accept it. A loader error found while parsing JSON also reports its byte offset in the diagnostic. The candidate engine composed from management documents is bounded by its inputs (a 256 KiB file, 128 documents of 4 KiB, 8,192 reputation rows) instead of by a 2 MiB arena, which a valid file could exhaust. Otherwise decisions, WAF findings, loader acceptance, diagnostics, rule hashes and quota scopes are those of the Zig library; a differential test replays generated requests, mutated policy files and management documents against its output.
2026-10-15 (Rust port): dynamic policies are read from the SQLite or PostgreSQL storage of SID 0011 instead of Zaxonlite, and a rebuilt engine is published as an owned value behind an `Arc` instead of into the spare read-copy-update slot (SID 0002, SID 0010). A rebuild that fails keeps the previous engine, as before.
2026-10-15 (Rust port, security fix): rules match the request path in the normal form of SID 0009 (revision of 2026-10-15), and that is the path the origin receives. Zig matched the path as sent and forwarded it verbatim, while origins route on a decoded and resolved path. With a rule denying `/admin/*`, the spellings `//admin/x`, `/./admin/x`, `/%61dmin/x`, `/admin%2fx` and `/admin;x/y` all reached the origin. With `^/login$` at 24 bits, `//login` was challenged at the default 16 bits and escaped the rule's quota. The built-in `well-known` ALLOW let `/.well-known/..;x/admin` reach `/admin` on Tomcat and Spring without a challenge. Every path criterion, the static bypass paths, rule quotas and the challenge requirement now read the normal path. So do the reserved-route match, the original URI of forward authorization, and the URL the interstitial reports. The pattern grammar is unchanged, but a pattern matches only paths in normal form, so patterns must be written in that form. That means one slash between segments, no dot segments, unreserved characters unescaped, and other escapes with upper-case hexadecimal digits (`/caf%C3%A9/*`). A pattern that holds `;`, `%2F`, `%5C` or `\` matches nothing. Matching stays case-sensitive: `/admin/*` does not match `/ADMIN/x`. An origin that folds case needs a rule for each spelling. A path above the root (`/static/../../etc/passwd`) is refused with 400 before evaluation, where Zig let the WAF deny it as `waf:path_traversal`.
2026-10-15 (Rust port, security fix): `RequestView` carries the client address parsed once by the daemon (`client_address`); reputation and CIDR criteria read it instead of re-parsing `client_ip`, which remains for incidents and logs.
2026-10-15 (Rust port, rule patterns): an engine compiles each rule pattern once, when the rule enters the engine. It keeps what the pattern's form needs: the exact text, the prefix, or, for the case-insensitive substring form, a case-folded needle with a Horspool shift table. Zig interpreted the pattern grammar on every request and rebuilt its substring search table each time. The grammar and every match are unchanged.

2. Abstract

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.

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.

3. 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.

4. 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.

> **Rust port note (2026-10-15)**
> Path criteria and the static bypass paths match the request path in the normal form of SID 0009: one slash between segments, no dot segments, unreserved characters unescaped and other escapes in upper case. Write patterns in that form. Matching is case-sensitive. See the revision note.

5. 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`.

6. 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
`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 to 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.

7. 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.

Rust port note (2026-10-15)
In the Zig trie an `ip_rules` entry for `0.0.0.0/0` (or `::ffff:0:0/96`) never matched. In the Rust port it matches every IPv4 client. See the revision note.

8. 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.

Rust port note (2026-10-15)
The Rust port reads these rows from SQLite (one node) or a shared PostgreSQL database (a cluster) as SID 0011 specifies, and publishes each rebuilt engine behind an `Arc` instead of a read-copy-update slot.

9. 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.

10. 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.

11. 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 to 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.

12. References

- SID 0004, *Semantic Attack Inspection and GCRA Rate Limiting: The Shield Surface*, the Semantic WAF of the Shield surface.
- SID 0005, *Distributed Storage Architecture: Zaxonlite Integration for Dynamic Policies, Replicated Reputation, and Incident Forensics*, the Zaxonlite `policies` table and its replication.
- `benchmarks/results/latest.json`, the measured engine memory and Gate/Shield timings in the Zig repository.

Lifecycle event

```
2026-10-15: prediscussion -> discussion. Ported from the Zig repository's
Typst record
```

Lifecycle event

```
2026-10-15: discussion -> published. State carried over from the Zig
repository's registry
```