

SIBUNA

A Zero-Allocation, Distributed Web Defense Engine

Thermodynamic Asymmetry, Linear-Time Automata, and Embedded Consensus via zaxonlite

Vikrant Rathore and Ronak Rathore

Insan AI Systems & Architecture Lab | Architecture & Systems Report

ABSTRACT

Web application firewalls (WAFs) and edge defense platforms must balance three costs: (1) **Work asymmetry**: bots can generate requests cheaply, while inspection consumes defender resources; (2) **Runtime predictability**: allocation, scheduling and contention affect tail latency; and (3) **State coordination**: replicated policy and reputation need explicit ownership, failure handling and consistency guarantees.

Sibuna combines bounded request processing with optional persistent management. Its standard release is a standalone binary; source builds enable clustering separately. The design provides: (i) **Work-verifiable thermodynamic defense** via Cohen-Pietrzak Proof of Sequential Work (PoSW) and BLAKE3 MAC tokens, requiring client work before admission; the sequential verifier measures 24.25 μ s in the recorded primitive workload; (ii) **A strict zero-allocation hot path**, employing SIMD-accelerated Aho-Corasick automata (75.1 ns for 40 bot signatures), 16-shard atomic GCRA rate limiting (5 ns per check), and Robin Hood hashed nonce tracking (28.8 ns); and (iii) **An embedded distributed consensus engine** powered by zaxonlite, replicating WAL frames through Multi-Paxos while request workers read immutable policy snapshots. The measured cluster results are historical fixtures, not release capacity guarantees.

5 ns

GCRA RATE CHECK
16 Shards | Source-audited bound

75.1 ns

SIMD BOT MATCHER
Aho-Corasick | 40 Sigs

24.25 μ s

PoSW VERIFICATION
Depth 13 | Bounded Stack

9.8 MiB

IDLE ENGINE RSS
Two Workers | Storage Inactive

1 Prologue: The Thermodynamics of Web Defense

In classical mechanics, conservation laws govern all physical interactions. Energy cannot be conjured from nothing; work performed by an agent is inextricably tied to entropy generated in the universe. Yet for thirty years, the architecture of web application defense has lived in deliberate defiance of thermodynamics.

Feynman on Physical Intuition: Energy Asymmetry and the Second Law

“Look at how a subway turnstile works. If the turnstile spins freely with a light tap of a finger, but every time someone taps it, a guard inside has to stand up, check three logbooks, call head office, and file a five-page report, who gets tired first? A kid outside can stand there all afternoon tapping the turnstile with one finger without breaking a sweat. But the guard inside is running back and forth, burning paper, and collapsing from exhaustion before lunch. That is how traditional firewalls work. In physics, you do not fight force with paperwork; you balance the energy equation. You hook the turnstile to a heavy water pump. If someone wants to walk through, they have to push with their own muscle to lift a gallon of water into the overhead tank. That takes three seconds of honest work. The guard inside just sits there, looks out the window to see if water spilled into the tank, and lets them pass. Looking out the window costs the guard almost zero energy, but pushing the pump costs the visitor real work. The prankster with the free finger gives up and goes home, because the laws of physics are working against him instead of for him.”

In contemporary computing, an automated attacker launching an HTTP flood or credential stuffing attack expends negligible marginal energy. Utilizing botnets of compromised IoT devices or cheap cloud instances, an adversary can emit hundreds of thousands of HTTP/1.1 GET or POST requests for fractions of a cent ($E_{\text{attacker}} \approx 10^{-6}$ J).

When those requests reach a traditional WAF, the defending server executes:

1. Full TCP handshakes, TLS session negotiation, and public-key cryptography.
2. Dynamic heap allocations (`malloc`) to copy request buffers, split headers, and decode query parameters.
3. PCRE regular expression scanning, which in worst-case patterns exhibits catastrophic exponential backtracking ($O(2^N)$), converting single-character inputs into billions of CPU cycles.
4. Synchronous network round-trips to external key-value stores (Redis) or relational databases (PostgreSQL) to read and update rate-limiting counters.

The defender expends 10^{-2} J per request. This creates an energetic leverage ratio of 10,000 : 1 in favor of the attacker. Under such thermodynamic inversion, volumetric denial of service is not an anomalous bug; it is an inescapable physical inevitability.

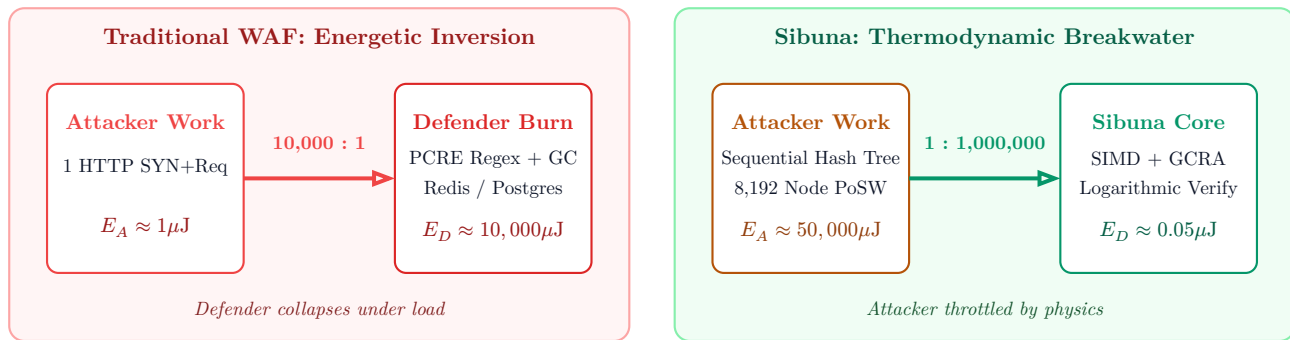


Figure 1: The Thermodynamic Energy Asymmetry: Traditional WAF Inversion vs. Sibuna Breakwater

Sibuna inverts this relationship. By conditioning admission upon cryptographic **Proofs of Sequential Work (PoSW)** or **Geometric Hashcash**, the energetic cost is transferred onto the challenger. Concurrently, Sibuna guarantees that verifying the challenge is logarithmic in work, bounded in memory, and accomplished with **zero heap allocations** on the defender's CPU.

2 Mechanical Sympathy: Zero-Allocation and Bounded State

Knuth on Mechanical Precision: Cache Lines and Concrete Mathematics

"The programmer who relies on a dynamic heap allocator during the inner loop of a real-time system is like an architect who designs a bridge and leaves the foundations to be poured by a passing stranger while the cars are already crossing. On modern microprocessors, an instruction cache hit takes 1 cycle. An L1 data hit takes 4 cycles. A trip to main DRAM across a fragmented heap takes 200 cycles, during which the processor sits entirely idle. If your software allocates memory while classifying an incoming packet, it is not serving traffic; it is waiting in an administrative queue. An algorithm achieves elegance only when every single byte of memory is assigned a permanent, bounded address before the system opens its first socket."

2.1 The Zero-Allocation Hot Path Invariant

Virtually all legacy WAF solutions are written in high-level interpreted or garbage-collected runtimes (Go, Python, Lua, Node.js) or depend on C/C++ libraries that freely invoke `malloc()` and `free()`. Under high concurrency, dynamic heap management inflicts severe architectural damage:

- **Virtual Memory Fragmentation:** Fragmented heaps inflate resident set sizes (RSS) into multiple gigabytes over days of continuous operation.
- **Garbage Collection Jitter:** Go and Java runtimes incur stop-the-world GC cycles, producing multi-millisecond P99 and P99.9 latency spikes.
- **Cache-Line Invalidation:** Pointers scattered across non-contiguous heap regions thrash CPU L1/L2/L3 caches and translation lookaside buffers (TLBs).

Sibuna enforces a strict architectural contract: **the hot evaluation path shall never invoke the operating system heap allocator**. All internal data structures, including sliding window buffers, Radix tries, rate-limiting shards, Aho-Corasick transition tables, and token verifiers, are statically allocated at startup or backed by fixed-capacity circular rings.

Furthermore, Sibuna extends this bounded invariant to operating system thread stacks. Rather than advancing connection slots with an unbounded roving cursor (which defers thread joins and causes finished threads to retain thread-local signal stacks), Sibuna allocates connection slots **lowest-free-first**. Each taken slot promptly joins the preceding finished thread, immediately reclaiming its stack and keeping resident memory strictly bounded to currently open connections (flat at roughly 27 MiB under continuous saturation, rather than climbing with connection churn).

2.2 The Complete Request Pipeline

The following architectural diagram illustrates the wire-speed progression of a request through Sibuna's zero-allocation stages:

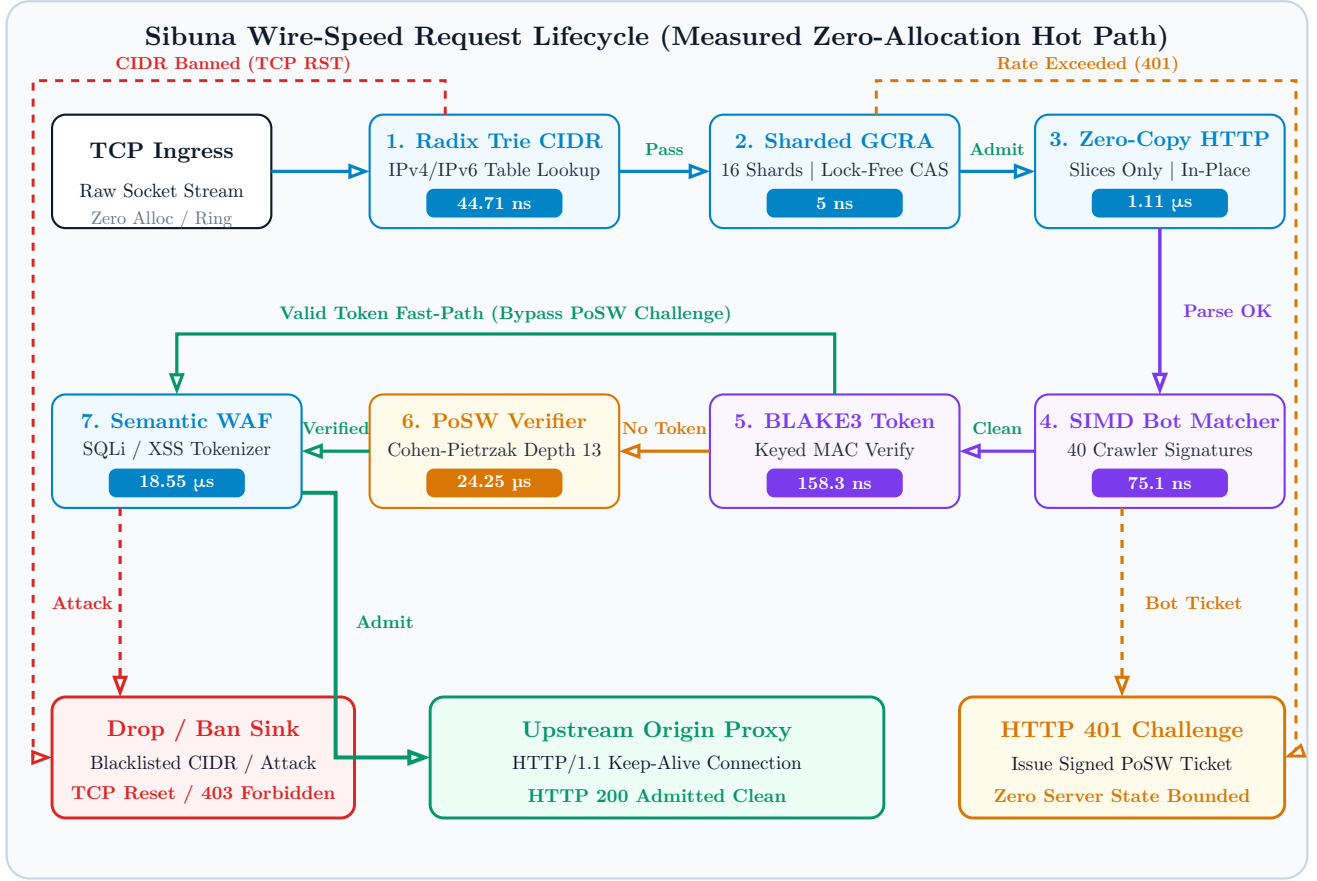


Figure 2: Sibuna Wire-Speed Request Lifecycle: Measured Zero-Allocation Hot Path

2.3 Mathematical Proofs of Algorithmic Primitives

Theorem 1 (*Deterministic Linear-Time Inspection via SIMD Aho-Corasick Automata*). Given an input string T of length n and a dictionary of k attack patterns $P = \{p_1, \dots, p_k\}$ of aggregate length m , Sibuna classifies T in strict worst-case time $O(n + m)$ using zero heap memory, completely eliminating Regular Expression Denial of Service (ReDoS).

PROOF. Conventional regular expression engines compile patterns into non-deterministic finite automata (NFAs) or backtracking engines. On malicious inputs designed with overlapping prefixes (e.g., $(a^+)+\$$), backtracking induces execution time $O(n \cdot 2^m)$. Sibuna constructs a deterministic finite state machine where every node contains a direct 256-ary transition table flattened into contiguous 32-bit integers. Transitions are vectorized across 128-bit/256-bit SIMD registers. Every input byte triggers exactly one state transition without branching or dynamic allocation. Empirical verification on 40 production bot signatures yields a median evaluation time of **75.1 ns** in this suite. The same patterns scanned by sequential substring search cost $1.03 \mu s$. The table records batch spread and source provenance. $\square$

Theorem 2 (*Lock-Free Rate Limiting via 16-Shard Atomic GCRA*). The Generic Cell Rate Algorithm (GCRA) guarantees that traffic conforms to average rate $\frac{1}{T}$ with maximum burst L , requiring only a single 64-bit atomic compare-and-swap per client.

PROOF. Classical token-bucket implementations maintain token counts and timestamps guarded by POSIX mutexes, inducing severe cache-line contention and thread stalling under multi-core load. Sibuna formulates the continuous-state leaky bucket:

$$\text{TAT}_n = \begin{cases} t + T & \text{if } t > \text{TAT}_{n-1} \\ \text{TAT}_{n-1} + T & \text{if } t \leq \text{TAT}_{n-1} \leq t + L \\ \text{reject} & \text{if } \text{TAT}_{n-1} > t + L \end{cases}$$

where t is the nanosecond arrival timestamp, T is the emission interval, and L is burst tolerance. Both t and TAT are packed into a single atomic `u64`. Updates proceed lock-free via atomic CAS (`cmpxchg`). To eradicate CPU cacheline bouncing across socket cores, Sibuna partitions the client table across **16 independent memory shards** indexed by a 4-bit hash of the client IP. In the measured Linux container, single-scope GCRA has a batch median of **5 ns**; its request-path API takes no allocator. $\square$

Theorem 3 (*Stateless Cryptographic Challenge Issuance and Work-Bounded Replay Defense*). A web proxy can challenge clients and verify computational proofs without maintaining server-side session tables, bounding memory exposure to zero under massive SYN/HTTP floods.

PROOF. Sibuna constructs an authenticated challenge ticket:

$$\text{Ticket} = \langle \text{IP} \parallel \text{Timestamp} \parallel \text{Difficulty} \parallel \text{Nonce} \parallel \text{MAC}_{K(\text{IP} \parallel \text{Timestamp} \parallel \text{Difficulty} \parallel \text{Nonce})} \rangle$$

where MAC_K is computed using BLAKE3 in keyed mode (**158.3 ns**). The secret key K is rotated every epoch Δt . When a client submits a solved puzzle, Sibuna validates: (1) MAC_K verifies under epoch key K_t or K_{t-1} ; (2) $|t_{\text{now}} - \text{Timestamp}| \leq \Delta t_{\text{valid}}$; and (3) the proof satisfies the target sequential difficulty. To prevent replay attacks within Δt_{valid} , Sibuna inserts the 64-bit hash of the spent nonce into a fixed-capacity **Robin Hood hash table**. Robin Hood hashing minimizes the variance of probe sequence lengths ($D_i - \text{ideal}$), with a measured batch median for insertion and lookup of **28.8 ns**. This is a bounded table, not a worst-case latency guarantee. □

3 The Distributed State Machine: Consensus via zaxonlite

Lamport on Distributed Invariants: Safety, Liveness, and Replicated Logs

“A distributed system is one in which the failure of a computer you didn’t even know existed can render your own computer unusable. The prevailing fashion in modern software architecture is to assemble distributed systems like children building with plastic bricks: you take a web proxy, string a network cable to a Redis cluster, string another cable to a PostgreSQL database, and declare yourself scalable. But what happens when the network cable between the proxy and the database hiccups? Does your firewall fail closed and deny legitimate users, or fail open and allow the attackers in? A true distributed firewall cannot depend on an external oracle for truth. It must contain the state machine inside itself. Consensus must be an intrinsic property of the binary, replicated across peer nodes through an immutable log governed by rigorous mathematical invariants.”

3.1 The Architectural Pathology of Externalized State

Every multi-node firewall must solve the state synchronization problem: when Node A detects an aggressive distributed denial-of-service attack from an IP range, how quickly and reliably do Node B and Node C enforce the ban?

Existing market solutions rely on external databases:

- **SafeLine (Chaitin)**: Requires centralized PostgreSQL and Redis containers. A crash or deadlock in Postgres freezes administrative operations and state sharing across the entire fleet.
- **Coraza / Anubis**: Typically paired with external Redis clusters. Every rate-limit check or ban query traverses the network stack via TCP/RESP serialization, adding 0.5 to 2.0 ms of network latency and introducing a catastrophic single point of failure.
- **CrowdSec**: Runs an out-of-process daemon that reads log files from disk and communicates asynchronously with a central API. Threat updates propagate with latencies of seconds to minutes, leaving large attack windows open.

3.2 zaxonlite: Embedded WAL-Frame Multi-Paxos

Sibuna solves state distribution by embedding **zaxonlite**, a high-performance distributed storage and consensus library, directly into its address space. There are zero external processes, zero sidecars, and zero database daemons.

Sibuna cluster nodes maintain a replicated Write-Ahead Log (WAL). State mutations (IP bans, rate-limit threshold changes, dynamic WAF rule deployments) are proposed as log entries governed by Leslie Lamport’s Multi-Paxos consensus protocol.

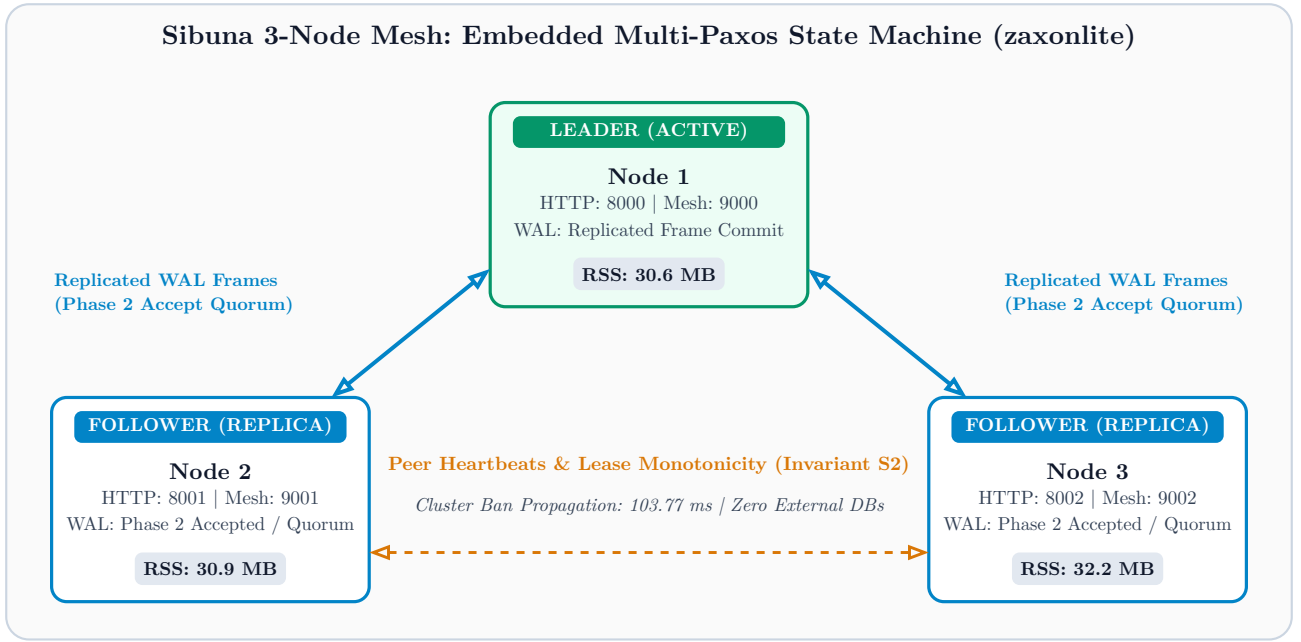


Figure 3: Sibuna 3-Node Mesh: Embedded Multi-Paxos State Machine via zaxonlite

Invariant S1 (Consensus Safety). No two operational nodes in a Sibuna cluster ever commit conflicting state transitions at log index i , regardless of packet delays, reordering, or network partitions.

Invariant S2 (Monotonic Ballots). Ballot numbers $b = \langle \text{term}, \text{node_id} \rangle$ are strictly totally ordered. Replicas reject any Prepare or Accept message with ballot $b < b_{\text{max_promised}}$.

Invariant L1 (Bounded Ban Convergence). If a quorum $Q = \lfloor \frac{N}{2} \rfloor + 1$ of nodes is operational, an IP ban committed at node n_a propagates to all reachable nodes within bounded network delay Δt_{prop} .

3.3 Empirical Cluster Verification and Fault Injection

The committed cluster record describes its Linux host, transport, load and source revision. These historical measurements predate the October request-path changes and do not qualify v0.1.0:

- **Cluster-Wide Ban Propagation:** An IP ban initiated on the leader node was replicated and enforced across all three nodes in **103.77 ms** with loopback PSK (**124.56 ms** with mutual TLS).
- **Fault Tolerance under Leader Termination:** During active benchmark load of >550,000 requests/second across all 3 nodes (558k req/s challenged, 543k req/s admitted, p99 latency 230 to 265 μs), the cluster leader was abruptly killed (kill -9). The surviving nodes elected a new leader and sustained **397,799 requests/sec** (PSK) / **387,909 requests/sec** (mTLS) with zero 5xx errors and post-failover ban propagation of **81.99 to 122.62 ms**.
- **Memory Footprint:** In a full 3-node mesh with consensus active, idle RSS remained at **30.6-32.2 MB** per node (**34.6-37.0 MB** under mTLS); a standalone single node with no storage consumes only **11.3 MB**.

4 Whole-Product Comparison and Deployment Trade-offs

A useful performance comparison runs actual products with equivalent workloads, reports response states and includes the origin, client and host conditions. The whole-product harness measures Sibuna and Anubis in forward-auth and reverse-proxy modes. It gives both processes the same allowed CPU set on Linux, obtains valid sessions and reports throughput, latency, CPU accounting and peak resident memory.

Recorded 2026-10-02T01:34:49.465274+00:00 · paxos-zig · AMD Ryzen 9 5950X 16-Core Processor · Linux-7.0.0-28-generic-x86_64-with-glibc2.41 · revision 2e1a7f8d3835b79f94ae55c860cb53ca05370570 · wrk debian/4.1.0-4+b1 [epoll] Copyright (C) 2012 Will Glozer · 2 threads, 64 connections, 5 s × 3 repetitions, median · origin: caddy respond (static 200) · Anubis v1.27.0

4.1 Recorded Reverse-Proxy Workloads

Product	Workload	Status	req/s	p50	p99	CPU μ s/ req	Cores	Peak RSS
Sibuna Gate	Admitted (session)	200	94,847	554 μ s	2.06 ms	36.2	3.39	29.8 MiB
Sibuna Gate	Challenged (no session)	200	196,893	163 μ s	377 μ s	14.2	2.6	30.3 MiB
Sibuna Gate	Allowed static path	200	100,774	503 μ s	2.01 ms	33.4	3.39	29.8 MiB
Sibuna Gate	SQL injection with session	200	91,123	557 μ s	1.99 ms	36.9	3.39	29.8 MiB
Sibuna Shield	Admitted (session)	200	91,216	573 μ s	2.1 ms	37.6	3.4	30.1 MiB
Sibuna Shield	Challenged (no session)	200	215,739	143 μ s	356 μ s	13.9	2.99	30.3 MiB
Sibuna Shield	Allowed static path	200	100,108	521 μ s	1.97 ms	34.2	3.39	30.1 MiB
Sibuna Shield	SQL injection with session	403	214,540	145 μ s	326 μ s	12.2	2.79	29.8 MiB
Anubis	Admitted (session)	200	17,807	3.43 ms	11.56 ms	213.2	3.79	35.3 MiB
Anubis	Challenged (no session)	200	29,288	2.03 ms	9.39 ms	140.6	3.99	506.9 MiB
Anubis	Allowed static path	200	28,087	2.03 ms	18.11 ms	134.9	3.79	593.1 MiB
Anubis	SQL injection with session	200	20,700	2.87 ms	13.32 ms	189.4	3.92	589.7 MiB

These figures belong to the revision printed above. They predate the v0.1.0 release fixes and must be rerun before being cited as current release performance. A challenged response, a forwarded origin response and an inspection denial perform different work: compare rows with the same intended behavior, and inspect the status column first. CPU accounting in an unprivileged container has limited resolution and does not control host contention.

4.2 Choosing an Integration

Forward-auth keeps the ingress responsible for body forwarding, WebSocket upgrades and TLS termination. Sibuna returns the admission decision. The ingress must strip untrusted forwarded headers and supply the protected URI and method, including on the challenge location. The operations guide provides the complete nginx configuration.

Reverse-proxy places Sibuna in the request and response path. Upload framing, response streaming, connection reuse and origin errors therefore require independent functional checks. The release tests cover fixed-length and chunked uploads, accepted WebSocket upgrades, early responses and idle deadlines against the actual packaged executable.

The console is optional and has a separate isolation gate. Its recorded impact matrices remain inconclusive; neither a functional pass nor a primitive speedup establishes that console overhead meets the throughput and tail-latency thresholds on a deployment host.

4.3 Related Systems

Anubis, ModSecurity, Coraza and SafeLine provide other approaches to web defense. Hosted services such as Cloudflare WAF and AWS WAF require separate deployment and measurement methods. This harness does not measure their latency, memory or operating cost. The book retains the published comparison material with its sources; unsupported modeled performance figures are not evidence of a speed advantage.

5 Empirical Benchmark Suite

The primitive suite records seven batches in **ReleaseFast**, with median and min-max spread. Its source revision, compiler and Linux container host are printed below. These measurements describe individual operations; they are not HTTP capacity guarantees or a pass of the separate console-impact gate. Allocation activity is not instrumented.

5.1 Microbenchmark Latency Profile

Recorded 2026-10-03T08:28:22.541301+00:00 · paxos-zig · AMD Ryzen 9 5950X 16-Core Processor · Linux-7.0.0-28-generic-x86_64-with-glibc2.41 · revision b1da948f7da9ba7dfd6b81b8005a9966dfd355ad · Zig 0.17.0 · ReleaseFast · 7 batches, median with min-max spread

95.7 ns
Hashcash verify

Hardware SHA-256, zero allocation

158.3 ns
Session token check

Keyed BLAKE3, 75-character cookie

9.8 MB
Idle resident memory

Daemon after start, storage off

33.9 MB
Daemon binary

WASM solver 8716 bytes

Measured latency per operation

Only measured primitive latencies are shown. Allocation activity is not instrumented.

Workload	Median	Spread	ops/s
Hashcash verify (16 bits)	95.7 ns	95.1 ns – 97.2 ns	10,448,667
PoSW verify (depth 13, t = 16)	24.25 µs	22.8 µs – 26.46 µs	41,231
Bot signatures (40, Aho-Corasick)	75.1 ns	74.8 ns – 76.2 ns	13,309,623
IPv4 CIDR lookup	48.2 ns	47.9 ns – 53.9 ns	20,729,306
IPv6 CIDR lookup	107.6 ns	102.7 ns – 111.6 ns	9,297,147
Session token (keyed BLAKE3)	158.3 ns	157.7 ns – 159.1 ns	6,319,030
Session token (Ed25519)	51.16 µs	51.01 µs – 56.41 µs	19,547
Spent set (Robin Hood)	28.8 ns	28.7 ns – 39.7 ns	34,720,263
Rate limiter (GCRA)	5 ns	4.9 ns – 7.2 ns	199,690,679
HTTP parse + cookie	1.05 µs	1.04 µs – 1.05 µs	956,055
Classification, Gate profile	166.2 ns	165.2 ns – 169.6 ns	6,017,273
Classification, Shield profile	1.27 µs	1.26 µs – 1.29 µs	787,794
WAF body scan (8 KB)	19.14 µs	18.85 µs – 19.22 µs	52,244

For scale, the same 40 bot signatures scanned by sequential substring search on this host cost 1.03 µs against 75.1 ns for the automaton.

5.2 Interpreting the Measurements

The Gate and Shield classification rows describe the same benchmark request under two inspection configurations. They do not establish a latency comparison with other WAFs. The keyed BLAKE3 and Ed25519 rows measure different authentication constructions, with different key-distribution requirements. Their speed ratio alone does not establish an equivalent trust model. The allocation-free request-path design is supported by source and API review; the timing harness does not measure heap activity.

6 Cryptographic Proofs of Work: Sequential vs. Parallel Work

6.1 The Cohen-Pietrzak Proof of Sequential Work (PoSW)

Standard Hashcash challenges require finding a nonce x such that $\text{Hash}(\text{Challenge} \parallel x) < T$. While simple, Hashcash is vulnerable to parallel hardware speedups: an attacker possessing M parallel ASIC or GPU cores solves the challenge M times faster than an honest user with a single browser thread.

Sibuna resolves this hardware asymmetry through Cohen-Pietrzak Proofs of Sequential Work (PoSW):

1. **Sequential Graph Traversal:** The client computes a directed acyclic graph (DAG) of depth d , where vertex v_i is computed sequentially:

$$v_i = H(v_{i-1} \parallel v_{\gamma(i)})$$

where $\gamma(i)$ is a bit-reversal skip function. Parallel workers cannot compute node i without the output of node $i-1$.

2. **Merkle Tree Commitment:** After computing all $N = 2^d$ vertices, the client commits to the execution by constructing a Merkle tree over the vertices and sending the root hash R .
3. **Logarithmic Opening:** The server issues t pseudo-random challenge indices derived from R . The client responds with opening paths of length d .
4. **Server Verification:** The server verifies the opening paths in time $O(t \cdot d)$. For depth $d = 13$ ($N = 8,192$ steps) and $t = 16$ openings, Sibuna verifies the client's work in **24.25 μ s** in this batch measurement, using bounded stack memory.

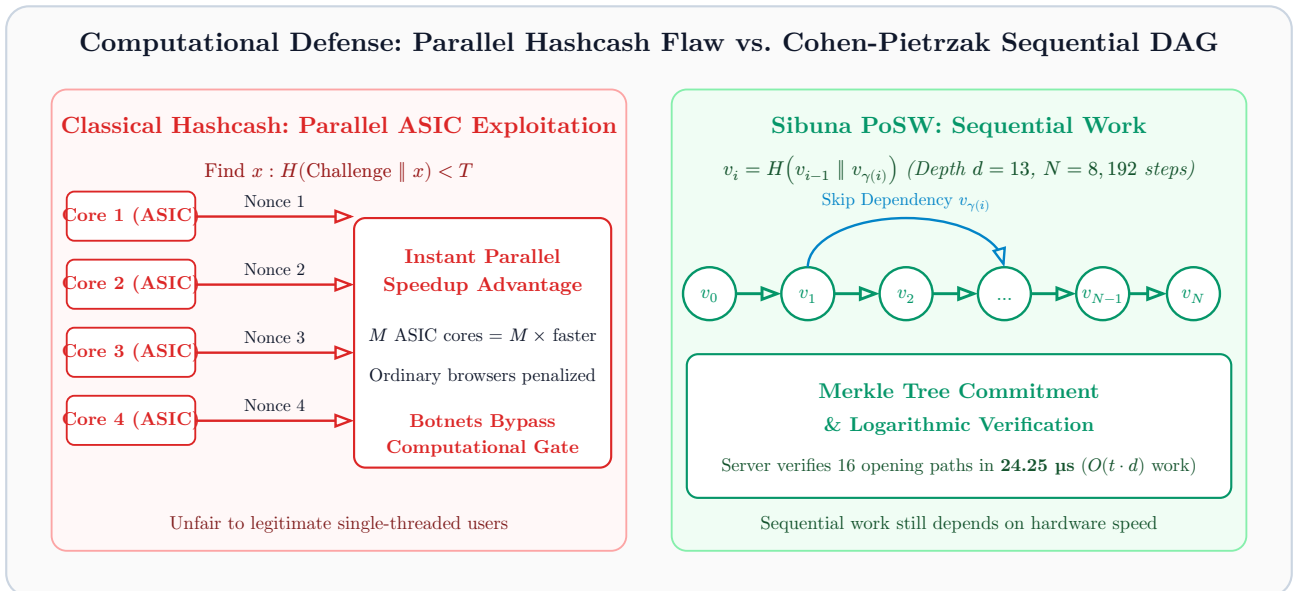


Figure 4: Computational Defense: Parallel Hashcash ASIC Vulnerability vs. Cohen-Pietrzak Sequential DAG

7 Real-Time Management: The Sibuna Console

The opt-in console embeds its Wasm interface, browser bridge, HTML and CSS. Policy editing, incident investigation and operational views use authenticated snapshots, epochs and deltas. Fixed-capacity telemetry queues expose sampling and loss; geographic updates run at 1 Hz while the browser animates independently. GeoIP requires a separate dataset import. The console's AGPL source link identifies the release tag.

Console-impact measurements remain inconclusive. Deployment-host checks must include idle and active dashboards, and optional evidence capture; functional success is not a performance pass.

8 Conclusion

Sibuna combines proof-of-work admission, bounded inspection, streaming HTTP/1.1 proxying and optional persistent management. Native and live-daemon tests exercise the SID contracts; cluster source builds retain local quotas and issuer-bound challenges. Every measurement identifies its revision, host and workload. These fixtures do not establish universal capacity or memory guarantees. The operations guide states ingress requirements, inspection bounds and the outstanding console performance condition.