SIBUNA

ゼロアロケーションで作業に基づく Web 防御エンジン

作業に基づく許可、線形時間の検査、有界な状態、SQL によるストレージ

Vikrant Rathore

協力：Ronak Rathore、Kanak Rathore
Insan AI | リリース 1.0.0-beta.3

概要

Web アプリケーションファイアウォールやエッジでの防御は、アーキテクチャ上の三つの問題を抱えがちです。一つ目はコストの逆転です。送信者はほとんど何も費やさずにリクエストを送れますが、防御側はそれを解析し、バックトラックするかもしれない正規表現で走査し、外部ストアに問い合わせます。二つ目は実行時の予測不能性です。リクエスト経路でのヒープ割り当て、ガベージコレクションによる停止、複数のコンテナーからなるデプロイによって、レイテンシー分布の裾を抑えることが難しくなります。三つ目は運用の重さです。禁止、トークン、制限のための共有状態は、運用者がファイアウォールと並べて動かさなければならないサービスに置かれることがよくあります。**Sibuna** は、少数の素朴な仕組みを一つの自己完結したバイナリにまとめてこれらの問題に取り組む試みであり、本稿では各仕組みの背後にある考え方を説明します。(i) 作業に基づく許可：ブラウザーは、ルールごとに選ばれる Hashcash パズルまたは Cohen-Pietrzak の逐次作業証明を解いてセッションを獲得し、サーバーは自ら保存していないチャレンジに対して、その解答を 70.2 ns または 22.3 us で検証します。(ii) 有界で割り当てを行わないリクエスト経路：バイトクラス Aho-Corasick オートマトンは 40 個のボットシグネチャを 55.7 ns で照合し、逐次探索の 13.7 倍の速さです。GCRA リミッターはクライアントあたり 16 バイトの状態で 6.2 ns で判断します。Robin Hood 使用済み集合は解答を 21.0 ns で記録します。SQL インジェクション、クロスサイトスクリプティング、コマンドインジェクション、パストラバーサルの検査も同じ走査の中で、入力長に比例する時間で行います。(iii) 独自の合意プロトコルを持たない共有状態：組み込みの SQLite ファイルが一つのノードを、共有する一つの PostgreSQL データベースがクラスターを支えます。どちらも一つのストレージスレッドが読み、リクエストがその完了を待つことはありません。1 台の AMD Ryzen 9 5950X で、デーモンに四つの CPU を割り当てた場合、Sibuna は forward-auth リクエストに毎秒約 200,000 件（ワークロード全体で 197,532 から 211,636 件）応答し、p99 は 305 から 327 us です。毎秒 100,000 リクエストのオープンループ負荷の下でも p99.9 を 62 us に保ち、常駐メモリの最大値は約 16 MiB です。各数値をどのように得たかを報告し、まだ測定していないことも列挙します。

6.2 ns	55.7 ns	22.3 µs	16 MiB
GCRA レート検査	**ボット照合器**	**PoSW の検証**	**デーモンの最大 RSS**
ハッシュ済み | 16 シャード	Aho-Corasick | 40 シグネチャ	深さ 13 | 16 開示	wrk 負荷時 | 4 CPU

AMD Ryzen 9 5950X の 1 コア（デーモンは 4 CPU）で動作する Sibuna、2026 年 10 月 15 日：benchmarks/results/primitives-latest.md と product-latest.md。方法は評価の節で説明します。

1 序章：リクエストの経済学

サーバーに届くリクエストはどれも、サーバーの注意を要求します。メソッド、パス、いくつかのヘッダーだけでは、そのリクエストに応える価値があるとは言えません。そこで防御は、リクエストとオリジンの間に判断を置きます。許可するか、拒否するか、まず検証可能な作業を送信者に求めるかです。本稿では、Sibuna におけるこの判断を、その数学からメモリ上のバイトまでたどります。

> **Feynman に学ぶ物理的直観：作業の代価を払うのは誰か**
>
> 軽く触れただけで回る回転ゲートを考えましょう。その後ろには守衛がいて、ゲートが回るたびに立ち上がり、三冊の台帳を調べ、本社に電話しなければなりません。外にいる子どもは午後じゅう苦もなくゲートを回し続けられますが、守衛は昼前に疲れ果ててしまいます。多くのファイアウォールはこのような配置になっています。安く済む側が攻撃者なのです。物理学はもっと良い配置を教えてくれます。回転ゲートをポンプにつなぎ、通りたい人はバケツ一杯の水をゲートの上のタンクまで汲み上げなければならないようにするのです。汲み上げるには少しの正直な労力が要りますが、守衛はタンクを一目見て水位が上がったことを確かめるだけです。一目見るのは守衛にとってほとんどただですが、汲み上げることは訪問者にとって本物のコストです。通勤者は一度汲み上げれば一日中通れますが、いたずら好きはすぐに家へ帰ります。

本稿の囲み記事は私たち自身の言葉であり、Richard Feynman、Donald Knuth、Leslie Lamport の説明の仕方にならって書いたものです。囲み記事が出典を明記している箇所を除き、引用ではありません。

自動化された送信者が一つのリクエストを送るためのコストはごくわずかです。リクエストが従来のファイアウォールに届くと、防御側は TLS ハンドシェイクを完了し、リクエストをコピーして分割するためにメモリを割り当て、一連の正規表現を適用し、カウンターを更新するために共有ストアとの間でネットワークの往復を行うことがあります。この非対称性に普遍的な比率を当てはめることはしません。そのようなものは存在しないからです。比率はオリジン、キャッシュ、攻撃によって変わります。代わりに、単純なモデルで判断を目に見える形にします。c_oを許可し



図 1 作業はどこに行くか。上は、先に検査する防御で、送信者が送るリクエストごとに自分の作業を費やします。下は、Sibuna が送信者に期待値2^b回のハッシュ試行を求め、その解答をハッシュ一回で検証する場合です。メーターはモデルですが、検証時間は測定値です（16 ビットでの Hashcash 検証、`benchmarks/results/primitives-latest.md`）。

たリクエストあたりのオリジンの作業量、c_gをリクエストあたりのゲートの作業量、aをゲートが許可するリクエストの割合とします。ゲートが元を取れるのは、次の式が成り立つときに限ります。

$$c_g < (1 - a)\, c_o,$$

つまり、ゲートがオリジンから省く作業がゲート自身のコストを上回るときです[1]。ここから二つの帰結が得られます。ゲート自身のコストc_gは、入力が何であれ小さく有界でなければなりません。また、許可の判断はサーバーではなく送信者にとって高価でなければなりません。

Sibuna は、許可の条件に作業証明を課すことでこの非対称性を逆転させます。セッションを獲得するために、ブラウザーは SHA-256 ダイジェストの先頭にb個のゼロビットが並ぶ nonce を探すか、逐次作業証明を計算します。探索の期待値は2^b回のハッシュ試行で、$b = 16$では 65,536 回です。サーバーが確かめるのは候補一つだけです。サーバーの検査は状態も持ちません。チャレンジの発行では何も書き込まず、メモリを占めるのは代価を支払って得た解答だけです。

作業証明は、人間であること、身元、意図を立証するものではありません。送信者はコンピューターを借りることも、セッションの有効期間内にそれを共有することも、作業を複数のマシンに分散することもできます。パズルが変えるのは許可の価格です。許可されたリクエストに何をさせるかは、引き続きポリシーと検査が決めます。

2 設計の概要：一度の走査で行う三つの判断

Sibuna は許可、検査、レピュテーションを三つの別々の判断として扱い、オリジンが何か作業をする前に、一つのプロセスの中で、リクエストに対する一度の走査でそれらを下します。これらは二つの製品の運用構成として提供します。*Gate* は、ルール、CIDR レピュテーション、ローカルな大量アクセス対策を備えた作業証明による許可を提供します。標準の *Shield* は、これにリクエストのペイロードに対する有界な検査を加えます。どちらの運用構成も、オリジンの前に置くリバースプロキシとして、または Caddy や Nginx などのイングレスが問い合わせる forward-auth エンドポイントとして動作します。ポリシー、レピュテーション、インシデント履歴の永続ストレージは、どちらでも選択できます。一つのノードのローカルに置くことも、クラスターで共有することもできます[2]。

[1] 技術ガイド第 I 部「許可は経済的な判断」。この不等式は処理能力のモデルであり、測定値ではありません。帯域幅、接続の状態、クライアントの遅延は省いています。

[2] SID 0002「Decision」、技術ガイド第 II 部「一つのバイナリに三つの考え方」、技術ガイド第 VI 部「宣言的ポリシーエンジン」。ルールごとの例、要件チケット、作業レベル付きのセッションは技術ガイド第 IV 部から、適応的な基準は技術ガイド第 IV 部と SID 0006（revision note、Part H）からとりました。

表 1 二つの製品の運用構成（SID 0002、技術ガイド第 II 部）

機能	Gate（--gate）	Shield（--shield、標準）
許可	Hashcash または逐次作業によるブラウザーチャレンジ。クライアントと、支払った作業に結び付いたセッショントークン	同じ
ポリシー	宣言的なルール（パス、User-Agent、ヘッダー、IPv4 と IPv6 の CIDR、信頼できるイングレスから受け取る JA4 TLS フィンガープリント）と ALLOW、DENY、CHALLENGE、WEIGH。ボットシグネチャ。レピュテーション	同じ
大量アクセス対策	GCRA によるレート制限、ハニーポット、一時的な禁止	同じ
ペイロードの検査	無効	SQL インジェクション、クロスサイトスクリプティング、コマンドインジェクション、パストラバーサル。ボディーの先頭 8 KiB が対象
任意の状態	SQLite に保存するポリシーとレピュテーション、または PostgreSQL を通じた共有	加えてインシデントのフォレンジック：全文検索とキャンペーンのクラスタリング

2.1 設計上の選択

以下の考え方の多くは研究文献や先行するシステムに由来し、その系譜は技術ガイドの第 II 部でたどっています。私たちが試みたのは、それらを一つの有界なリクエスト経路に組み合わせ、各ステップを安価に保ち、結果を誠実に測定することです。以下に、それぞれの選択とその理由を挙げます。

1. 発行にサーバーのコストがかからないチャレンジ。チャレンジは鍵付きハッシュで封をした自己認証型のレコードなので、発行しても状態を書き込みません。メモリを占めるのは代価を支払って得た解答だけで、それもチャレンジの期限が切れると消えます（アルゴリズムとその保証）。
2. ルールごとに選ぶ二つの作業層。Sibuna は Hashcash に加えて、Cohen-Pietrzak の逐次作業証明を提供します。その健全性は量子的な証明者に対しても証明されており、証明の検証はマイクロ秒単位で終わります。ブログが 16 ビットの Hashcash を求める一方で、決済のルートは 20 作業ビットの逐次作業を求める、といった使い分けができます（作業証明：並列探索と逐次作業）。
3. 支払った作業を記憶するセッション。セッショントークンは、保有者が解いた方式と作業ビットを記録します。トークンが許可するのは、それ以上を要求しないルートだけなので、安価なセッションで高価なルートを開くことはできません。
4. 対称鍵のトークン。セッションは鍵付き BLAKE3 で封をします。検証は 155.8 ns で、Ed25519 署名の約 260 分の 1 の作業量です。また、量子探索による高速化も二次的なものにとどまります。
5. セッションで回避できない検査。許可と検査は同じ走査の中で行い、有効なセッションが満たすのはチャレンジだけです。セッションを持ち、`q=' OR 1=1--`を含むリクエストは、やはり拒否されます。
6. 線形時間で構造を理解する検査。一回のバイトクラス走査、一回のオートマトン走査、小さな構造トークナイザーによって、バックトラックしうる正規表現を使わずに、入力長に比例する時間で判断します。
7. 鍵付きの配置による有界な状態。どのテーブルも容量が固定され、満杯になったときの振る舞いが定められています。キーは鍵付きハッシュで配置するので、攻撃者が特定の相手を狙って衝突を起こすことはできません（鍵付きハッシュとセキュリティレビュー）。
8. リクエストごとの割り当てがなく、停止のないポリシー更新。接続はすべての判断を一つの固定バッファーから行い、新しいポリシーエンジンはリクエストも書き手も待たせずに公開します（メカニカルシンパシー：ゼロアロケーションと有界な状態）。
9. 負荷に追従する難易度。毎秒 50 件の検証済み解答という基準を超えると、レートが 2 倍になるごとに作業ビットを 1 つ、最大 6 つまで加えます。こうして大量アクセスは指数的に多くの代価を払い、静かなトラフィックは基本のコストのままです。
10. インシデントから見つけるキャンペーン。拒否したリクエストは小さな埋め込みベクトルとともに記録し、類似したインシデントをキャンペーンにまとめます。運用者はそれをコンソールから調査できます。

3 メカニカルシンパシー：ゼロアロケーションと有界な状態

> **Knuth に学ぶ機械的な精密さ：キャッシュラインと具体数学**
> リアルタイムシステムの内側のループで汎用のアロケーターを呼ぶプログラマーは、設計の一部を見知らぬ他人に委ねたことになります。アロケーターは平均すれば速いかもしれませんが、そのコストはヒープの履歴に依存し、その履歴を書くのはリクエストを送ってくる誰かです。何かを最適化する前に、まず数えるべきです。各ステップが何バイトに触れるか、キャッシュラインをいくつ使うか、依存するロードが何回あるか。それから測定すべきです。数え方も間違うことがあるからです。プログラムが優美であるとは、必要なバイトのすべてに最初のソケットを開く前から決まった置き場所があり、各ループがその数学が要求するちょうどの作業だけを行うことです。

3.1 ゼロアロケーションの不変条件

リクエスト経路での動的な割り当てはどれも、サービス拒否攻撃のてこになります。数日にわたる断片化、アロケーターのロックでの競合、割り当てを誘うように作られたリクエストによる増幅です。Sibuna の規則は、分類、検証、プロキシへの受け渡しをヒープ割り当てなしで行うことです。接続は、ソケットが続く間、64 KiB の読み込みバッファーと 16 KiB の書き込みバッファーを持ち、さらに接続の開始時に一度だけ割り当てる 40 KiB のプロキシ用作業領域を持ちます。それ以外はすべてこれらのスライスです。パーサーはリクエストのメソッド、パス、クエリ、ヘッダー、Cookie を読み込みバッファーの中で特定し、どれもコピーしません[3]。実装の型システムは、解析済みのリクエストが、それが記述するバイトより長く生存できないことを保証します。そのため、リクエストがまだ読んでいる間にバッファーを再充填することはできません。

この規則はリクエスト経路の契約であり、プロセス全体についての主張ではありません。スレッドの生成、起動処理、ストレージは割り当てを行います。この契約は、割り当てを数えるテストで確認しています。デーモンの定常状態のテストは割り当てを数え、プロキシのテストは 500 回のキープアライブのプロキシリクエストで割り当てがないことを確かめます[4]。

3.2 有界で再利用されるスレッド

受け付けた接続はそれぞれ専用のスレッドで処理するので、遅いクライアントが遅らせるのは自分自身だけであり、リクエスト経路は同期的なままです。イベントループも検討しましたが、スレッドモデルを残しました。リクエスト経路の速さは、ゼロコピーのパーサー、接続ごとの固定バッファー、割り当てを行わない判断から来ており、そのどれもイベントループでは改善しないからです[5]。接続スレッドの数は`--max-connections`（標準は 1,024）で上限が決まり、それを超えると受け付けループは何も生成せずに `503` で応答します。一つのソケットのピアアドレスが持てる接続は標準で 64 までで、IPv6 のピアは/64 ネットワーク単位で数えます。

スレッドは、join 可能なスロットからなる固定のテーブルから取り出します。終了したスレッドは join されるまでスタックを保持するので、新しいタスクはカーソルを進めるのではなく、終了済みのスロットのうち最も番号の小さいものを join して使います。こうすると常駐メモリは、処理した接続の数ではなく、開いている接続の数に従います。接続の処理を終えたスレッドは待機し、次の接続を引き受けます。この最後の規則を加えることになった経緯は、測定が変えたことの節で説明します。

3.3 リクエストのパイプライン

リクエストは以下の検査を決まった順序で通ります。安価な検査が先に来て、リクエストを終わらせる判断は、より高価な段階が動く前に下されます。

有効なセッションが満たすのはチャレンジです。拒否や WAF の検出を覆すことは決してありません。プロキシのほうは、オリジンの応答ヘッダーを読み、そのフレーミング（ボディーなし、長さ指定、chunked 符号化、切断まで）を分類し、ボディーを正確に中継し、きれいな境界で終わったオリジンのソケットを 256 本のストライプ化されたプールに返します。そのため、プロキシしたクライアントの接続は開いたままになり、リクエストごとにオリジンへ再接続することはありません。

3.4 リクエストを止めずにポリシーを公開する

ポリシーが変わると、ポリシーエンジンはリクエスト経路の外で再構築されます。リクエストが作りかけのエンジンを見ることがあってはなりません。ストレージスレッドは完全なエンジンを構築し、参照カウント付きで、世代カウンターを持つスロットに公開します。各接続はキャッシュを持ち、acquire ロード一回で世代を読み込み、公開があったときにだけ新しい参照を取得します。古いエンジンは、それを保持する最後のキャッシュが手放したときに解放され、書き手が読み手の退出を待つことはありません。競合の下では、キャッシュからの読み込みは読み書きロックを取るより 400 倍以上安価です。エンジン自体は約 200 KiB（209,052 バイト）のコンパクトなテーブルを持ち、128 個のルールと 8,192 個のトライノードを収容できます。

[3]技術ガイド第 V 部「接続ごとに一つのスレッド、ただし上限付き」と「接続ごとに一つのバッファー」。
[4]技術ガイド第 V 部「接続のループ」、`cmd/sibuna/tests/steady_state.rs` と `crates/net/tests/steady_state.rs`。
[5]SID 0010「Concurrency model」と「Rationale and alternatives」、SID 0002「Request contract」とその revision notes（アドレスごとの接続数、IPv6 の接頭辞、ストライプ化されたオリジンのプール）。

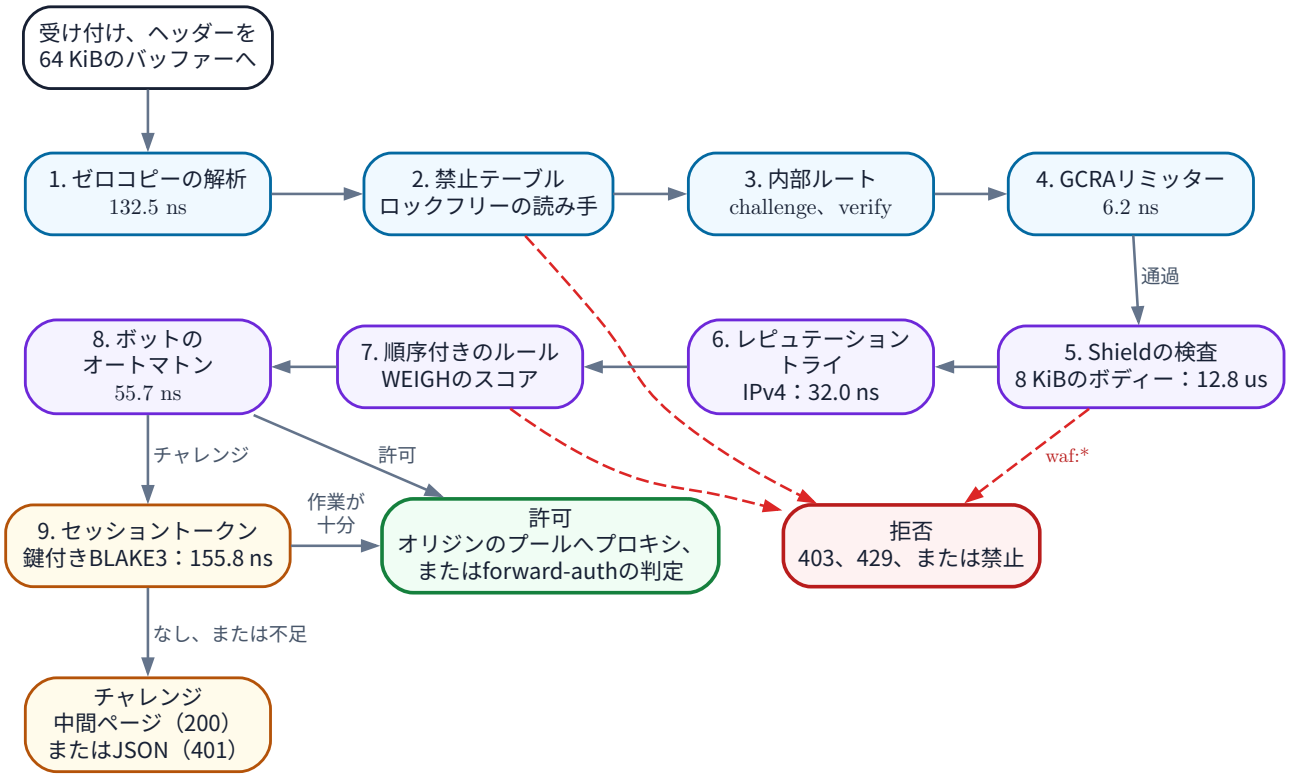


図 2 リクエストのパイプラインと、プリミティブのスイートが測定している各プリミティブの 1 コアでの中央値です（benchmarks/results/primitives-latest.md）。各段階は番号の順に動きます。破線の矢印は早期の拒否を示します（GCRA リミッターとレピュテーショントライも同じように拒否します）。ブラウザーのリクエスト一つの分類全体にかかる時間は、Gate のプロファイルで 75.9 ns、Shield のプロファイルで 790.8 ns です。

3.5 シャードごとのロック

接続スレッドが共有するテーブル（使用済み集合、リミッター、禁止テーブル）は、キーのハッシュによって 16 個のシャードに分割され、各シャードはそれぞれ専用のキャッシュラインに置かれます。クリティカルセクションは数十命令の長さなので、スピンロックと、短くスピンしてから眠るアダプティブミューテックスを比較測定しました。競合がなければスピンロックは約 1 ナノ秒節約しますが、エッジサーバーは CPU より多くのスレッドを動かします。保持者がプリエンプトされたスピンロックでは、待つ側のすべてがタイムスライスを空費します。競合の下ではミューテックスが勝ったので、そちらを採用しました。

4 アルゴリズムとその保証

リクエスト経路上の各プリミティブには、何にどれだけのコストがかかり、何を保証するかを述べた命題があります。そのうち三つを定理として、論証の概略とともに示します。完全な証明、定義、参考文献は、SID 0006 と、Sibuna 技術ガイドの第 III 部から第 VI 部にあります。

表 2 1 スレッドと 16 スレッドがループで読み込むときのエンジン読み込み一回のコスト。Ryzen 9 5950X（技術ガイド第 V 部、2026 年 10 月）

読み手	1 スレッド	16 スレッド
読み書きロック、読み込みガードのみ	4.2 ns	752 ns
読み書きロックと新しい参照	7.6 ns	833 ns
キャッシュした参照による世代の確認	0.8 ns	1.8 ns

表 3 シャードのロックのスループット（毎秒百万操作）。Ryzen 9 5950X（技術ガイド第 V 部、SID 0010）

ワークロード	アダプティブミューテックス	スピンロック
使用済み集合、16 スレッド	10.8	8.6
使用済み集合、64 スレッド	11.2	6.9
六つのホットなアドレスに対する GCRA、16 スレッド	44.9	19.3
六つのホットなアドレスに対する GCRA、64 スレッド	37.4	6.5

定理 1（バイトクラス Aho-Corasick オートマトンによる線形時間の検査）． $\mathcal{P}$を総長mのパターンの集合とし、そのバイトは$c \le 256$個のクラスに分かれるものとします。また、テキストの長さをMとします。Sibuna はオートマトンを$O(mc)$の時間と空間で構築し、いずれかのパターンがテキストに現れるかどうかを$O(M)$時間で報告します。入力 1 バイトあたりの依存するテーブル読み込みは一回で、パターンの数には依存しません。どのような入力もバックトラックを引き起こせません。

証明の概略。Aho と Corasick のオートマトンは、失敗リンク付きのトライを構築した後、テキストを一度だけ走査します。Sibuna は構築時に失敗リンクを遷移表に組み込むので、入力 1 バイトは遷移一回であり、失敗の連鎖をたどることはありません。バイトクラスとは、いずれかのパターンに現れるバイト（文字の大文字と小文字は一つのクラスにまとめます）か、それ以外のすべてのバイトからなる単一のクラスです。そのため表の 1 行は 256 ではなくc個の項目を持ちます。一致状態は他のすべての状態の後に番号を付けるので、一致の検出は比較一回です。

測定。40 個のボットシグネチャに対して、オートマトンは 55.7 ns かかります。同じパターンと「いずれかに一致」の意味論による逐次の部分文字列探索は 763.0 ns で、13.7 倍の速さです。バイトクラスを使うと、ボットシグネチャの表全体は 42,244 バイトです[6]。

□

定理 2（GCRA の区間上限）． ウィンドウWあたりNリクエストという制限を固定し、送出間隔を$T = \max(1, \lceil W/N \rceil)$、バースト許容量を$\tau = (N-1)T$とします。クライアントは理論到着時刻TATを一つ持ちます。時刻tの到着が適合するのは$\mathrm{TAT} \le t + \tau$のとき、かつそのときに限り、その後$\mathrm{TAT} \leftarrow \max(\mathrm{TAT}, t) + T$とします。このとき、適合する到着で始まる長さ$L$の任意の区間に含まれる適合する到着は、高々$N + \lfloor L/T \rfloor$個です。

証明の概略。適合する到着に$1, \ldots, k$と番号を付け、その時刻を$t_1 \le \ldots \le t_k$とします。適合する到着はそれぞれ TAT を少なくともTだけ増やすので、j番目の後では$\mathrm{TAT}_j \ge t_1 + jT$です。到着$k$は適合したので、$\mathrm{TAT}_{k-1} \le t_k + \tau \le t_1 + L + \tau$です。これらを合わせると$(k-1)T \le L + (N-1)T$、すなわち$k \le N + L/T$が得られ、$k$は整数です。

実装。クライアントの状態は 16 バイトのセル一つで、リクエストごとの履歴も、固定ウィンドウの境界効果もありません。16 個のシャードは`--rate-table-cells`個のセル（標準は 262,144 個、4 MiB）を共有し、セルは鍵付きハッシュで配置します。空になったセルは、掃除用のスレッドなしにその場で再利用します。デーモンがすでにハッシュを計算したクライアントの検査は 6.2 ns です[7]。

□

定理 3（状態は支払われた作業に比例してのみ増える）． Sをデーモンが記憶するチャレンジのタグの集合とします。タグがSに入るのは、そのチャレンジに対する有効な証明が受理された後だけで、チャレンジの期限が切れると出ていきます。チャレンジ一つの有効期間における、受理された異なる解答のレートの上限をrとすると、$|S| \le r \cdot \mathrm{TTL}$です。取得されたものの解かれなかったチャレンジは、メモリを占めません。

証明の概略。チャレンジの識別子は自己認証型のレコードです。バージョン、アルゴリズム、難易度、開示数、発行時刻、クライアントの鍵付きフィンガープリント、擬似ランダム関数で導いた nonce、そのチャレンジを要求したルールのハッシュからなる 36 バイトに、16 バイトの鍵付き BLAKE3 タグが続き、URL 安全な base64 で 70 文字になります。発行しても何も書き込みません。検証では、まずタグを定数時間で確認し、次に経過時間とクライアントへの結び付き、さらに証明を確認し、その後で初めてタグを使用済み集合に挿入します。鍵付きモードの BLAKE3 を擬似ランダム関数としてモデル化すると、q回の検証クエリーを行う攻撃者がタグを偽造できる確率は、高々$q \cdot 2^{-128}$に PRF アドバンテージを加えたものです。

実装。使用済み集合は、Robin Hood オープンアドレス法による 16 個のシャードです。期限切れのエントリーをその場で上書きしてよいのは、ホームスロットから少なくとも同じだけ離れたキーだけです。そうしないと、後の検索が早く止まり、有効なタグを見落とすおそれがあります。この規則があれば掃除用のスレッドは不要で、拒否された挿入が有効なタグを追い出すこともありません。使用と検索を合わせて 21.0 ns です[8]。

□

4.1 意味解析ファイアウォール

Shield は、パス、クエリ、User-Agent、構造的でないヘッダー、ボディーの先頭 8 KiB を検査します。その最初の版から、私たちは教訓を得ました。最初の版は部分文字列の平坦なリストをすべてのヘッダーと照合していましたが、そのリストには/*が含まれており、どのブラウザーも`Accept: text/html,...,*/*;q=0.8`を送ります。本物のブラウザーがす

[6] `benchmarks/results/primitives-latest.md`（55.7 ns と 763.0 ns）。表のサイズは技術ガイド第 VIII 部「フットプリント」に示した`primitives-latest.json`の値。SID 0006 の Part F と、バイトクラスに関するその revision note。

[7] SID 0006 の Part D（Definition 22、Theorem 23）、SID 0004 の revision notes（鍵付きの配置、`--rate-table-cells`）、`benchmarks/results/primitives-latest.md`（`gcra_check_hashed`）。

[8] 技術ガイド第 III 部（補題 3 と補題 4、ワイヤーフォーマット）、技術ガイド第 V 部「Robin Hood 使用済み集合」、SID 0006 の Part E（Lemma 28）、`benchmarks/results/primitives-latest.md`（`robin_hood_spend_and_lookup`）。

べてSQL インジェクションに分類されてしまったのです。それ以来、検出器は構造に対してだけ反応してよいことにし、完全な Chrome のリクエストをエンジンに通して、何も検出されないことを要求するテストを設けています[9]。検査は四つの段階からなり、どれもフィールドの長さに比例する時間で動きます。

1. バイトクラスによる事前走査。256 項目の表による一回の走査で、どのクラスのバイトが現れるかを記録します。引用符、=、<、シェルの区切り文字、%、NUL、正規化のきっかけです。後続の検出器はすべて再走査せずにこのクラスを参照するので、引用符も等号もないフィールドが SQL トークナイザーに入ることはありません。
2. タグ付きのシグネチャのオートマトン。すべてのカテゴリーの強いシグネチャは、一つのオートマトンにまとめてあります（定理 1）。一致すれば、`waf:sqli`、`waf:xss`、`waf:path-traversal`、`waf:rce` のいずれかの名前で終端の拒否となります。
3. 構造トークナイザー。SQL トークナイザーは短い単語をそれぞれキーワード表で調べ、`or` と `and` の後の `literal = literal` 形式の恒真式を確認します。それ以外の場合は引用符が必須で、証拠が小さなスコアに達しなければなりません。「it's a group order from the shop」という語句のスコアは 3 で、通過します。クロスサイトスクリプティングと判定するには、スクリプトを持てるタグと、属性の位置にあるイベントハンドラー属性の両方が必要です。コマンドインジェクションと判定するには、区切り文字の後に、代入ではないコマンド名が続かなければなりません。そのため `a=1&id=5` はパラメーターであり、コマンドではありません。
4. 正規化。パーセントエスケープ、コメントの開始記号、JSON や HTML のエスケープ、まとめられる空白を含むフィールドはデコードしてから再び検査します。そのため `1%27/**/UnIoN/**/SeLeCt` や `%252e%252e/` も、元のシグネチャに帰着します。

8 KiB のボディーの走査は 12,784.3 ns で、1 バイトあたり約 1.56 ns です。運用者はカテゴリーを強制する代わりに監査とすることができ、その場合、検出結果は記録されますが対処はされません。このときフィールドはカテゴリーごとに判定しなければならず、監査した検出結果が強制する検出結果を隠してはなりません。Sibuna はフィールドをカテゴリーごとに一回ずつ走査するのではなく、シグネチャが現れるカテゴリーの集合を求めるために一回だけ走査し、その中から必要なカテゴリーだけを探します。SQL インジェクションを監査とした同じ 8 KiB のボディーでは 15,485.4 ns です[10]。Shield はあくまで有界なヒューリスティックの検査器であり、完全な SQL パーサーや HTML パーサーではありません。また、検査の範囲を超えたバイトは検査しません。

4.2 イングレスから受け取る TLS フィンガープリント

User-Agent は、どのクライアントでも好きなように書ける文字列です。JA4 フィンガープリントは偽造がより困難です。これは、クライアントが TLS 接続をどのように開くか、すなわちプロトコル、バージョン、暗号スイート、拡張、最初の ALPN 値を要約したものです。これらは、最初の HTTP のバイトが送られる前に、クライアントの TLSライブラリが決めてしまいます。ハンドシェイクを見られるのは TLS を終端するコンポーネントだけなので、JA4 を計算できるのもそのコンポーネントだけです。Sibuna は受信側の TLS を終端しないので、JA4 を計算しません。信頼できるイングレスが計算し、`--ja4-header` で指定したリクエストヘッダーで転送したフィンガープリントを読みます[11]。

- 信頼。このヘッダーは、転送されたヘッダーを信頼する場合にだけ読みます。その判断は `X-Forwarded-For` を扱う判断と同じです。信頼しないピアからのヘッダーは無視し、リバースプロキシモードでは、リクエストがオリジンに届く前にそのヘッダーのすべての写しを削除します。
- 検証。値を受け入れるのは 36 バイトの JA4 形式の場合だけで、しかもその名前のフィールドがちょうど一つあるときに限ります。形式が不正な値や重複した値は、存在しないものとして扱い、件数を数えます。
- ポリシー。ルールはフィンガープリント全体か、*が続く接頭辞と照合します。フィンガープリントのないリクエストが、このようなルールに一致することはありません。照合はバイト比較であり、ウォーム状態のリクエスト経路は依然としてヒープ割り当てを行いません。
- 記録。インシデントにはそのリクエストのフィンガープリントを保存し、コンソールはそれをインシデントの証拠として表示し、それでイベントを絞り込めます。禁止とレピュテーションのキーはアドレスのままです。多くの正当なクライアントが同じ TLS ライブラリを共有しているからです。

4.3 鍵付きハッシュとセキュリティレビュー

2026 年 10 月に行ったリクエスト経路の敵対的レビューで、解決済みと考えていた仕組みに弱点が見つかりました。私たちはそれぞれを修正し、その修正を各仕組みの設計記録に記しました[12]。

- 予測可能なテーブル配置。リミッターと禁止テーブルは、ソースに書かれたシードによるハッシュでキーを配置していたため、誰でも特定の相手と同じ探索窓を共有するアドレスを計算できました。現在の配置では、オペレーティングシステムの乱数から得た鍵による SipHash-1-3 を使い、テーブルごとに独自の鍵を導出します。デーモン

[9] SID 0004「Revision note (2026-09-07)」と「Verification」、技術ガイド第 VI 部「誤検知の教訓」。
[10] `benchmarks/results/primitives-latest.md`（`8kb_body_semantic_scan`、`8kb_body_sqli_audit_other_enforce`）、SID 0004 の revision note「inspection modes」、技術ガイド第 VIII 部「WAF の検査」。
[11] SID 0014「Specification」と「Security and operational considerations」、技術ガイド第 IX 部「イングレスから受け取る JA4 フィンガープリント」、`.github/release-notes.txt`。
[12] SID 0002 の revision notes（転送された身元、Cookie の値）、SID 0004 の revision notes（鍵付きの配置、禁止テーブル、測定したコスト、ゼロ初期化したテーブル）、SID 0006 の revision notes（使用済み集合の容量、適応的な制御器の入力）、SID 0010 の review history（セキュリティレビュー、2026-10-15）、`benchmarks/results/primitives-latest.md`（`gcra_check` と `gcra_check_hashed`）。

はリクエストごとに一度だけクライアントをハッシュし（8.7 ns）、そのハッシュをすべてのテーブルに渡します。リクエストのストア経路は 43 ns から 26 ns に短縮されました。
- 禁止の追い出し。有効な禁止が追い出されることはありません。キーが入りうるすべてのスロットに有効な禁止がある場合、新しい禁止は拒否されて件数が数えられます。そのため、禁止されたクライアントが自分の禁止を追い出すことはできません。
- 転送された身元。`X-Forwarded-For` の最も左のエントリーは、クライアントが書けてしまいます。Sibuna は右から `--trusted-hops` 番目の位置にあるエントリー、つまり最も近い信頼できるプロキシが追加したものを採用します。
- *Cookie tossing*。リクエストはセッション Cookie の値を複数持つことがあります。デーモンは最大八つまで試すので、兄弟ドメインが仕込んだ Cookie によって訪問者をチャレンジのループに閉じ込めることはできません。
- 使用済み集合の容量。65,536 個のタグのテーブルは、5 分間のチャレンジ有効期間の中で毎秒約 220 件の検証済み解答によって使い切られ、その後は誠実な解答が拒否されていました。現在、容量は設定可能で、標準は 1,048,576 個のタグ（24 MiB）です。
- 適応的な難易度。負荷の下で作業量を引き上げる制御器は、発行したチャレンジではなく検証済みの解答を数えるようになりました。チャレンジの発行は、大量アクセスを送る側にとって何のコストもかからないからです。

鍵付きの配置には代価があり、私たちはそれをテーブルごとではなくリクエストごとに一回だけ支払います。生のクライアントのバイトを自らハッシュするリミッターの検査は 12.8 ns かかります。デーモンが使う経路である、ハッシュ済みのクライアントに対する検査は 6.2 ns です。リミッターと使用済み集合はゼロ初期化した状態で割り当てるので、使われていないテーブルは常駐メモリにならず、設定した容量は使われるまでメモリを消費しません。

5 作業証明：並列探索と逐次作業

5.1 Hashcash の層

第一層は、ビット単位の粒度を持つ Hashcash です。チャレンジ文字列Cと難易度bに対して、クライアントは SHA-256 $(C \,\|\, \texttt{":"} \,\|\, \mathrm{dec}\,(N))$の先頭に$b$個のゼロビットが並ぶような nonce$N$を探します。nonce を十進数で表すことで、WebAssembly のソルバー、JavaScript による代替実装、ネイティブの検証器の間でワイヤーフォーマットが同一に保たれます。理想的なダイジェストでは、試行回数Kは$p = 2^{-b}$の幾何分布に従います。

$$P(K > k) = (1-p)^k, \qquad E[K] = 2^b, \qquad \mathrm{median}\,(K) \approx 2^b \ln 2.$$

探索には二つの限界があります。分散が大きいので、一回の解答が遅くても、ワーカーが壊れているとは限りません。また、独立した試行は並列に実行できるので、期待される総作業量は同じでも、コアの多い送信者ほど早く終わります[13]。

5.2 逐次作業の層

第二層は、Cohen と Pietrzak の逐次作業証明（EUROCRYPT 2018）に従います。そのラベルは深さnの完全二分木をなします。内部ノードは二つの子をハッシュします。葉は、根からの経路が右に曲がるところごとに、祖先の左の兄弟をハッシュします。この左の兄弟への辺が計算を逐次的にします。葉は、その左にあるすべての部分木が完成するまでラベルを付けられないので、根のラベルは$2^{n+1}-1$個のラベルすべてに順に依存します。根φを計算した後、証明者は Fiat-Shamir によってt個の葉を導き、それぞれについて葉のラベルと、その経路に沿ったn個の兄弟のラベルを送ります。検証者は開示された各葉を再計算してφまでハッシュをたどり、そのコストは固定の作業領域での$O(tn)$回のラベル計算です。

パラメーターは、数値が小さく保たれるように選んであります。作業ビットは$n = b - 3$によって深さに対応し、$[4, 24]$の範囲に制限されます。証明の大きさは$32(1 + t(n+1))$バイトで、深さ 13、16 個の開示では$32(1 + 16 \times 14) = 7200$バイトです。証明器は木の上位の段だけを保持し、開示しなければならない部分木を再計算するので、証明を含む作業領域全体は 100 KB 未満に収まります。ブラウザーのタブとネイティブのテストは同じ証明器を使います[14]。Blocki、Lee、Zhou（ITC 2021）は、量子ランダムオラクルモデルにおいて、この構成が量子的な証明者に対しても健全であることを証明しました。

これが何を立証するのかには注意が必要です。安全性の結果は、各論文が述べる構成と上限について、その論文のランダムオラクルの仮定の下で成り立ちます。私たちの具体的な SHA-256 のプロトコル、そのパラメーターの対応付け、ブラウザーでの所要時間を独立に監査したものではありません。初等的なサンプリングの上限$(1-\alpha)^t$、すなわちt個の独立な開示がすべて、固定した不正な割合αを外す確率は、開示を増やすと効果がある理由を説明します。しかし、それだけでは多くの根を総当たりする証明者に対する証明にはなりません。不正が決して安上がりにならないと無条件に主張することはしません。

5.3 往復とセッション

有効なセッションを持たないナビゲーションは中間ページを受け取ります。このページは要件チケットを持っています。これは、この判断が要求するチャレンジ（アルゴリズム、作業ビット、開示数）とルールのハッシュを、このクラ

[13] 技術ガイド第 III 部「Hashcash：第一層の数学」、技術ガイド第 I 部「確率変数としてのハッシュ探索」。
[14] 技術ガイド第 III 部「逐次作業証明：第二層」「証明者のメモリ」「パラメーターは所要時間ではない」、SID 0006 の Part B と、2026-09-07 のその実装レビュー。

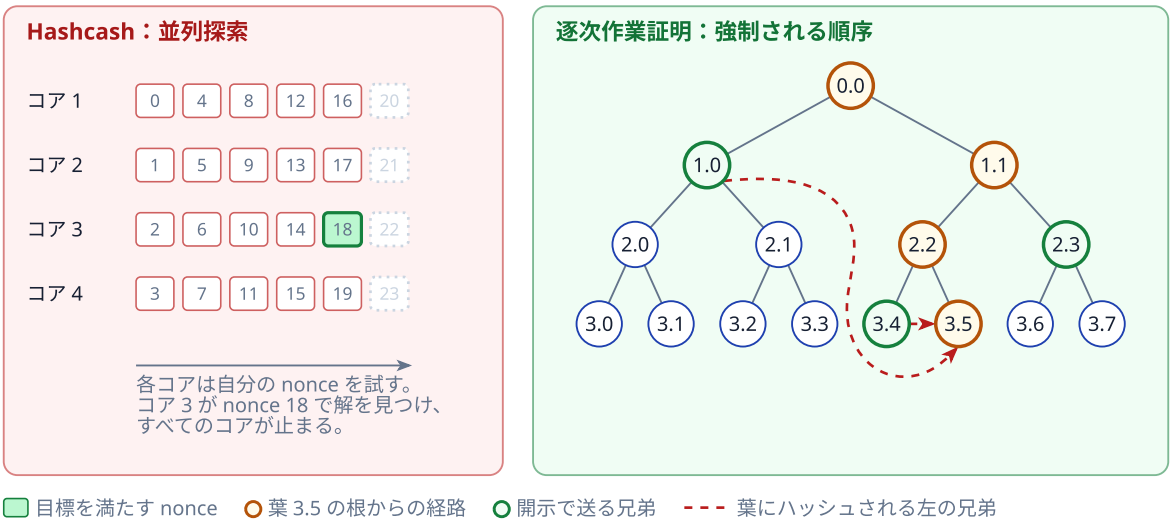


図 3 並列探索と逐次作業の対比。左は Hashcash で、nonce の試行は互いに独立なので、M個のコアは約M倍早く終わります。右は深さ 3 の Cohen-Pietrzak の木で、破線の左兄弟の辺のため、葉 3.5 はその左の部分木を待たなければなりません。葉 3.5 の開示では、そのラベルと緑色の三つの兄弟を送ります（技術ガイド第 III 部による）。

イアント用に封をしたもので、チャレンジ一つの有効期間だけ有効です。ページはこのチケットを使ってチャレンジを取得するので、作業量はそれを要求した判断によって固定されます。Web Worker が WebAssembly の証明器で、あるいは WebAssembly を使えない環境では JavaScript の代替実装でチャレンジを解き、解答を送信します。成功すると、サーバーは 75 文字のセッション Cookie を設定します。この Cookie は鍵付きフィンガープリントによってクライアントのアドレスと User-Agent に結び付いており、その後ページは再読み込みされます。ブラウザーの証明器はサーバーの検証器と同じソースからコンパイルされるので、両者の実装がずれることはありません[15]。

5.4 検証のコスト

どちらの層も SHA-256 が律速であり、Sibuna はプロセッサーにある場合は SHA 命令を実行時に選択するので、一つのバイナリがその命令を持つどのマシンでも高速に動きます。16 ビットでの Hashcash の検証は 70.2 ns、深さ 13 で 16 個の開示を持つ証明の検証は 22,272.4 ns です。ブラウザーの証明器は 14,744 バイトの WebAssembly モジュールです。エンドツーエンドのブラウザー検査では、ヘッドレス Chromium が中間ページのチャレンジを、WebAssembly で 29 から 34 ms、JavaScript の代替実装で 84 から 91 ms で解きました[16]。プリミティブのスイートが測定するのは検証だけであり、ネイティブでの所要時間はブラウザーでの解答時間を表すものではありません。

セッションには対称鍵のトークンを使います。検証者は発行者自身か、一つのシードを共有するクラスターなので、鍵付き擬似ランダム関数がちょうど必要な認証を提供します。鍵付き BLAKE3 のトークンの検証は 155.8 ns、Ed25519 のトークンは 40,587.5 ns で、署名の検証一回は MAC の検証約 260 回に相当します。この選択にはもう一つ理由があります。Shor のアルゴリズムは Ed25519 の基礎にある離散対数を破りますが、鍵付きハッシュに対して量子探索が与える高速化は高々二次的です。トークンを発行できない検証者を必要とするデプロイのために、Ed25519 も選択肢として残しています。

6 独自の合意プロトコルを持たない共有状態

> **Lamport に学ぶ分散不変条件：安全性、活性、ログの持ち主**
>
> 1987 年、Leslie Lamport は分散システムを、存在することさえ知らなかったコンピューターの故障によって、自分のコンピューターが使えなくなりうるシステムだと表現しました。これを助言として読めば、私たちが依存するコンピューターを数え、そのどれかが故障したときに何が成り立たなければならないかを述べよ、ということです。述べるべきことは二種類で足ります。安全性：悪いことは決して起こらない。たとえば、コミットされたどのトランザクションも書いていないポリシーに基づいてノードが動作することは決してない。活性：良いことはいずれ起こる。たとえば、データベースに到達できるノードは、いずれ最新のポリシーで処理する。合意プロトコルは、この両方を得るための美しい方法です。しかし、それは抱え込むには大きなものでもあり、ファイアウォールにとっての問題は、合意が良いものかどうかではなく、誰がログを保守すべきかです。

[15]技術ガイド第 IV 部「チャレンジの往復」と「セッション Cookie」、SID 0010「The browser solver」。

[16]`benchmarks/results/primitives-latest.md`。WebAssembly のサイズは技術ガイド第 VIII 部「フットプリント」に示した `primitives-latest.json` の値。ブラウザーでの解答時間は SID 0010 の criterion 3（Chromium 155、`cargo xtask browser`）から。SHA 命令の実行時の検出は SID 0006 の revision note（Parts A to C）から。

6.1 なぜ SQL なのか、なぜ独自の合意プロトコルではないのか

Sibuna が保存するものはわずかです。数百行のポリシー、レピュテーションのテーブル、そして運用者が他のシステムにもエクスポートするフォレンジック用のインシデントです。この状態には、専用の複製エンジンは必要ありません。必要なのは信頼できる SQL ストアで、一つのノードではローカルに、多数のノードでは共有されるものです。そのため、単一のノードは状態をデータディレクトリ内の組み込み SQLite ファイルに WAL モードで保存し、クラスターではすべてのノードが一つの PostgreSQL データベースを参照します[17]。

私たちは、ファイアウォールの内部で合意プロトコルを動かさないことを選びました。合意エンジンは、独自のフォーマットと独自の故障モードを持つ大きなシステムであり、ファイアウォールと並べて抱え込むには重く、シャットダウンやアップグレードのたびに作業を増やします。クラスターの状態の可用性と複製は、代わりに PostgreSQL が担います。PostgreSQL のストリーミングレプリケーションとフェイルオーバーのツール、あるいはマネージドサービスがそれを提供します。これはより謙虚な選択だと考えています。PostgreSQL のレプリケーションは、私たちのデプロイよりはるかに多くのデプロイで鍛えられています。また、Sibuna の書き込みレート（まとめて書くインシデントとまれなポリシー変更）は十分に小さいので、プライマリー一つがボトルネックになることはありません。

ストレージは任意のままです。`--data-dir` も `--database-url` も指定しなければ、デーモンはポリシーをファイルから読むインメモリのファイアウォールです。

6.2 一つのストレージスレッド

データベース接続は一つのストレージスレッドが所有し、それ以外が SQL に触れることはありません。リクエストスレッドがこのスレッドを待つことはありません。リクエストスレッドはインシデントのレコードを 512 スロットの有界なリングに入れ（満杯のときはレコードを捨てて件数を数えます）、公開されたエンジンを読みます。ストレージスレッドは、標準では 500 ms ごとに一回のティックを実行します。キューにある最大 32 件のインシデントを一つのトランザクションで書き込み、ポリシーのリビジョンを読み、変わっていればエンジンを再構築して公開します。再構築では読み込みの前後でリビジョンを読み、その間に書き手がコミットしていれば候補を破棄し、古いエンジンを使い続けます。運用者はポリシーのテーブルに直接書き込んでもかまいません。トリガーが書き手のトランザクションの中でリビジョンを進め、PostgreSQL では `NOTIFY` によってノードがすぐにティックを実行できます。ただし、保証を与えるのはあくまでポーリングです[18]。

クラスターの各ノードは、1 から 8,388,607 までの互いに異なる `--cluster-node` 識別子を持ちます。チャレンジの鍵は発行したノードに結び付いているので、解答はそのチャレンジを発行したノードに戻らなければなりません。一方、セッショントークンと要件チケットは、シードを共有するすべてのノードで有効です。インシデントの識別子は上位ビットにノードを持ち、各ノードはバッチと同じトランザクションの中で進める永続的な受領記録を保持します。そのため、結果が曖昧な障害の後でバッチを再実行しても、何かが二重に書き込まれることはありません。

> **不変条件：ストレージ層（SID 0011）。** **S1（所有者は一つ）。** SQL はすべてストレージスレッドで実行し、リクエストスレッドがデータベースを待つことは決してありません。**S2（完全なエンジン）。** リクエストが読むエンジンはすべて、公開される前に、ポリシーテーブルの一貫した一つのリビジョンから完全に構築されています。**L1（いずれ公開される）。** ポリシーテーブルへのコミット済みの変更は、データベースに到達できるすべてのノードに、次のティックまでに届きます。**F1（穏やかな損失）。** データベースが停止しているときは、最後に公開されたエンジンが処理を続け、インシデントは有界なリングにたまり、満杯になるとカウンターを増やして捨てられ、ストレージスレッドはバックオフしながら再試行します。

6.3 型付きのスキーマ

スキーマは、プログラムの型が述べることを SQL で述べるので、データベースはストレージ層が表現できない行を拒否します。SQLite のテーブルは `STRICT` です。列挙型の列には、プログラムが書く名前をちょうど列挙した `CHECK` があり、範囲のある整数には範囲の検査があります。また、列が `NOT NULL` であるのは、値が決して欠けない場合に限ります。どの文も値をパラメーターとしてバインドするので、引用符で囲んで SQL を組み立てることはありません。コンソールのストレージも同じ規則に従います。33 個の型付きテーブルを持つ一つの基準マイグレーションがあり、トリガーはありません。「トークンへの変更はすべて監査行を書く」といったコンソールの各不変条件は、SQLite でも PostgreSQL でも、文を一つのトランザクションで実行するちょうど一つの関数によって強制されます[19]。

デーモンを `postgres-tls` フィーチャー付きでビルドすると、PostgreSQL への接続は、URL の指定どおりに libpq の `sslmode` と `sslrootcert` で保護されます。このフィーチャーのないビルドは、起動時に `require`、`verify-ca`、`verify-full` を拒否し、平文に戻ることは決してありません。本番のクラスターでは `sslmode=verify-full` を使います。

似たインシデントはキャンペーンにまとめます。各インシデントは 64 次元の埋め込みベクトルを持ち、256 バイトとして保存されます。ストレージスレッドは全ノードの直近 16,384 個の埋め込みベクトルをメモリ上（4 MiB）に保持し、最近傍とのコサイン距離が 0.35 以下であれば、インシデントをその最近傍のキャンペーンに割り当てます。より

[17] SID 0011「Abstract」「Motivation and scope」「Why PostgreSQL for clusters」。以前の複製方式の設計は SID 0005 に記述があります。
[18] SID 0011「The storage thread」「Policy revision」「Incidents and campaigns」。
[19] SID 0011「Schema and migrations」とその review history（通信の安全性）、SID 0013「Abstract」。

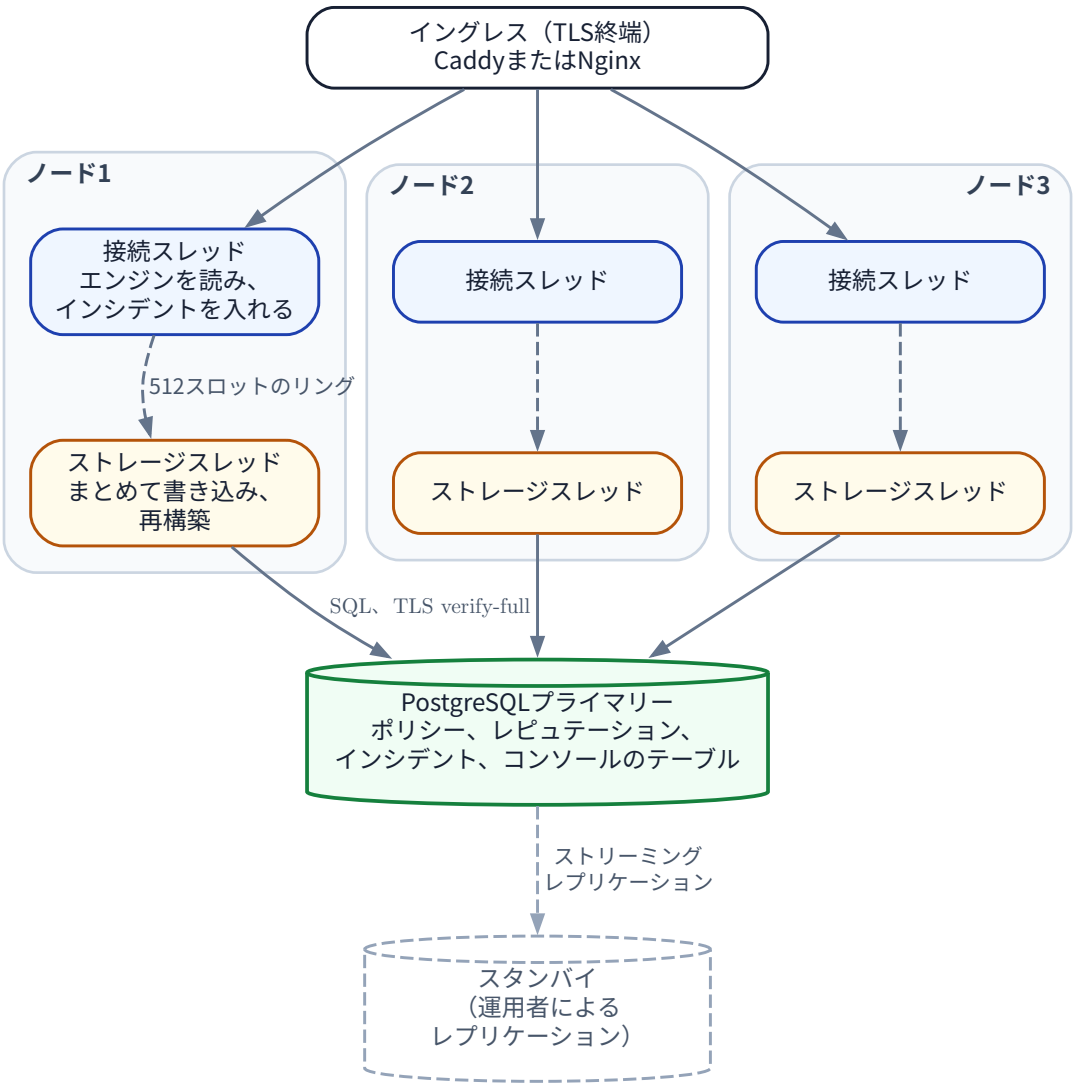


図 4 Sibuna のクラスター。各ノードはリクエスト経路をメモリ上に保ち、SQL はすべて一つのストレージスレッドに任せます。ノードは一つの PostgreSQL データベースを共有し、その複製とフェイルオーバーは PostgreSQL が担います（SID 0011）。単一のノードでは、代わりに組み込みの SQLite ファイルを使います。

古いインシデントとのクラスタリングは意図的に省いています。キャンペーンは本来最近のものであり、この上限によってメモリと作業量が一定に保たれるからです。

6.4 この設計がしないこと

この選択の限界ははっきり述べておくべきでしょう。Sibuna は独自の合意プロトコルを持たず、クラスターの一貫性は PostgreSQL に依存します。データベースとの接続を失ったノードは、再接続するまで最後のポリシーを使い続けます。あるノードで発行したセッションはすべてのノードで受け入れられ、あるノードで使用済みになった解答は、別のノードで再送しても拒否されます。ノードが一つのシードを共有し、チャレンジの鍵が発行したノードに結び付いているからであり、どちらの性質もデータベースを必要としません。SID 0011 は Committed です。一つの PostgreSQL データベース上の三つのノードによる機能テストで、あるノードで挿入したポリシーが一回のティックのうちに三つすべてで適用されること、あるノードでのハニーポットによる禁止が他のノードに届くこと、インシデントの識別子が決して衝突しないことを確かめています。ただし、一つの PostgreSQL データベース上の複数のノードに負荷をかけるハーネスはまだないので、本稿ではクラスターの測定結果を報告していません。リリース版の実行ファイルは PostgreSQL のサポートなしでビルドされています。クラスターでは `postgres-tls` フィーチャー付きで Sibuna をソースからビルドします[20]。

[20] SID 0011「Security and operational considerations」と「Acceptance criteria and implementation status」（criterion 2、`crates/storage/tests/multi_node.rs`）、`.github/release-notes.txt` の「Known limitations」、SID 0002「Authentication and work」（ノードに結び付いたチャレンジの鍵）、技術ガイド第 VIII 部「測定していないもの」。

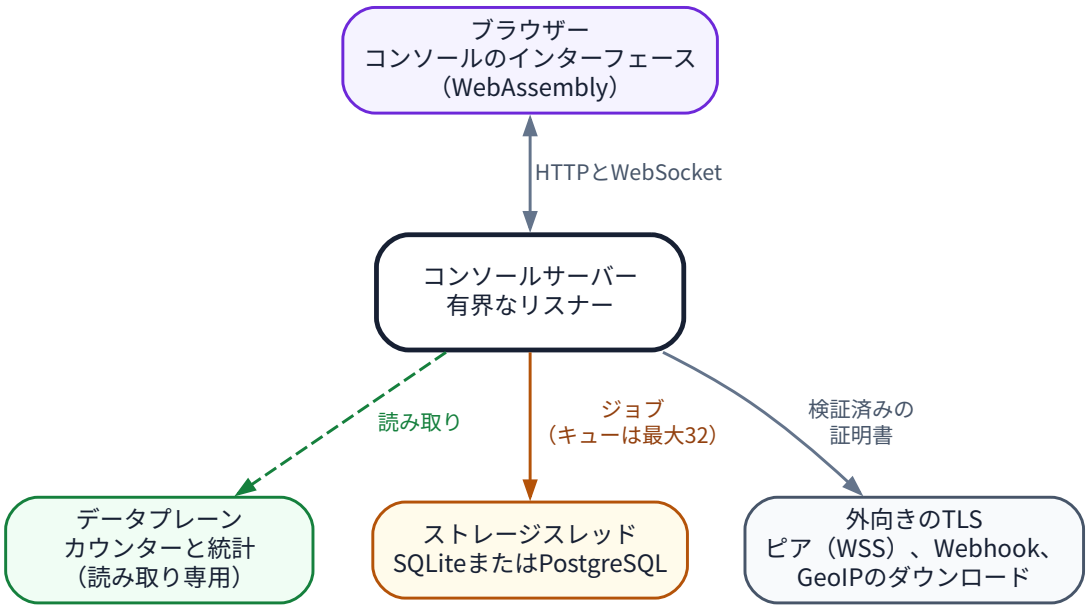


図 5 コンソール。ブラウザーはインターフェースを WebAssembly モジュールとして実行します。コンソールサーバーはデータプレーンのカウンターを読み、SQL をストレージスレッドに任せます。ピア、Webhook、GeoIP の配布元への外向きの接続には、検証済みの証明書による TLS を使います（SID 0007、0012、0013）。

7 コンソール

Sibuna には、明示的に有効にする管理コンソールが含まれています。ライブのトラフィックを表示する認証付きのダッシュボード、拒否から記録された証拠までをたどる調査の流れ、ポリシーの編集、アドレスグループ、オープンな GeoIP データによる国情報の付加、Argon2id のパスワードと TOTP の第二要素を使うユーザー、監査証跡、Webhook、クラスターの管理を備えています。コンソールは同じバイナリにコンパイルされ、そのインターフェースは WebAssembly モジュールとしてブラウザーで動作します[21]。

コンソールを定義づける制約は、リクエスト経路からの分離です。コンソールはデータプレーンのカウンターとデータベースを読み、データベースに書き込むことはできますが、リクエストスレッドに入ることも、そこで割り当てを行うことも、接続スレッドが必要とするロックを保持することも決してありません。そのリスナーが受け付ける接続は最大 96 で、そのうち 16 は HTTP 用に予約され、64 をブラウザーのストリーム、16 をピアのストリームに使えます。1.0.0-beta.3 では、インターフェースのモジュール全体は 724,330 バイト（gzip 圧縮で 272,568 バイト）です。これはビルドが強制する上限の 768 KiB を下回りますが、SID 0007 がレビューのために記録するしきい値の 640 KiB は上回っています[22]。

コンソールは、長時間続くピアのストリーム、Webhook、GeoIP のダウンロードのために外向きの TLS 接続を行います。証明書はシステムのルート証明書か運用者の CA ファイルに対して検証され、ホスト名の検証は無効にできません。平文の HTTP はリテラルのループバックアドレスに対してだけ許可され、証明書のエラーで平文に戻ることは決してありません。Sibuna は依然として受信側の TLS を終端しません。それは引き続きイングレスの役割です。

SID 0007 はコンソールの性能目標を定めています。コンソールを有効にしたときのコストは、データプレーンのスループットの 1% 以下、p99 レイテンシーの 10% 以下でなければなりません。測定結果は一つあります。コンソールを組み込んだうえで無効にした場合、ストレージのみの場合、有効にしてアイドル状態の場合のいずれでも、製品全体の 16 個のワークロードは、コンソールなしでビルドしたデーモンとの差が約 1% 以内にとどまり、これはノイズの範囲内です。負荷の下で八つのダッシュボードを動かす正式な判定はまだ実施していないので、目標を満たしたとは主張しません。八つのライブのダッシュボードによる機能的な実行は合格しました。

[21] SID 0007「Abstract」とその revision notes（リスナーの受け付け、インターフェースのモジュール、GeoIP とパスワード）、SID 0010 の review history（コンソールの移植、TOTP）、SID 0012、SID 0013。分離の測定は SID 0010 の review history（コンソールの移植）にあります。技術ガイド第 VIII 部「コンソールのオーバーヘッド」は、この記録がその結果ファイルを示していないことを注記しています。

[22] SID 0007 の review history（Rust への移植、統合、デーモンへの組み込み）。SID 0014 の JA4 フィールドが入る前の 719,406 バイトを記録しています。1.0.0-beta.3 のサイズは、デーモンのビルドスクリプトと同じ方法（`cargo rustc --lib -p sibuna-console-ui --target wasm32-unknown-unknown --profile wasm -- -C link-arg=--compress-relocations`）でビルドしたモジュールを `gzip -9` で圧縮したものです。

表 4 コンソール付きの一つの SQLite ノードで八つの Traffic ダッシュボードを表示、ヘッドレス Chromium 155（benchmarks/results/console-8-dashboards-2026-10-15.md、2026-10-15 記録）。結果：PASS

測定項目	値
データプレーンのリクエスト	30.0 秒で 6,000 件（毎秒 200 件、リクエストごとに 1 接続）
応答あり	6,000: 200 x 1,200, 401 x 3,000, 403 x 600, 404 x 600, 500 x 600
エラー（応答なし）	0
レイテンシー p50 / p90 / p99 / 最大	246 / 311 / 373 / 552 us（接続を含む）
デーモンの常駐メモリ：開始時、8 ダッシュボード表示時、トラフィック後	36,152 KiB; 38,028 KiB; 41,496 KiB
ダッシュボード	管理者セッション 3 つとウォールディスプレイ 5 つ。それぞれ 134 から 139 個の stats 差分を受け取り、同じ合計値で終了しました。例外、コンソールのエラー、Content-Security-Policy 違反はありませんでした

8 実証的な評価

8.1 方法

Sibuna の測定には、cargo xtask bench で実行する三つのハーネスを使います。プリミティブのスイートは、時間を計らないウォームアップの後、リクエスト経路の各操作を七つのバッチで計時し、変更可能な状態はタイマーの外で作り直し、操作ごとにバッチの中央値を、最速と最遅のバッチとともに報告します。製品全体のハーネスは、wrk（2 スレッド、64 接続、5 秒、3 回反復）でデーモンを駆動します。forward-auth モードとリバースプロキシモード、Gate と Shield の運用構成で、四つのワークロードを使います。admitted（有効なセッション）、challenged（セッションなし）、allowed_static（/robots.txt）、attack（セッションに加えて q=' OR 1=1--を含み、Shield だけが拒否する）です。レイテンシーのハーネスは、デーモンの振る舞いにかかわらず一定のレートで送信するオープンループの生成器を使い、各リクエストを送信予定の時刻から計時します。こうすることで、デーモン内での待ち行列の時間はデーモンの負担として計上されます[23]。実行は 2026 年 10 月 15 日に 1 台のホストで記録しました。Linux 7.0.0-28 で動作する AMD Ryzen 9 5950X（16 コア、32 スレッド）です。デーモンは四つのワーカーとともに CPU 0 から 3 で、負荷生成器とスタブのオリジンは CPU 8 から 15（もう一方の CCD）で、プリミティブのスイートは CPU 4 で動かしました。デーモンは標準のリリースビルドで、SQLite とコンソールを組み込んだうえで、ストレージを使わずメモリ上で動かしました。

測定の範囲. どの実行も、1 台のホストでループバックを使っています。TLS も広域ネットワークも本物のブラウザーも使っていません。ハーネスは割り当てを数えません。リクエスト経路で割り当てを行わないことは、代わりに定常状態のテストで確認しています。ハーネスはリリース版のバイナリを変更せずに実行し、その SHA-256 で識別します。このホストでの順位は、別のホストでの順位ではありません。

8.2 プリミティブのレイテンシー

表 5 Ryzen 9 5950X の 1 コアにおける Sibuna のプリミティブの所要時間（1 操作あたりのナノ秒）。七つのバッチの中央値と、最速および最遅のバッチ（benchmarks/results/primitives-latest.md、2026-10-15T22:34:59Z 生成）

サブシステム	ワークロード	中央値	最小-最大
作業証明	Hashcash の検証、16 ビット	70.2	69.9-71.0
作業証明	PoSW の検証、深さ 13、16 開示	22,272.4	22,257.0-22,316.4
ボット照合器	Aho-Corasick、40 シグネチャ	55.7	55.3-57.2
ボット照合器	逐次の部分文字列探索、40 シグネチャ（参考）	763.0	762.2-763.5
IP 分類器	IPv4 CIDR の検索	32.0	31.8-32.2
IP 分類器	IPv6 CIDR の検索	79.3	78.5-79.6
トークン	鍵付き BLAKE3 のセッショントークン	155.8	155.3-156.4
トークン	Ed25519 のセッショントークン	40,587.5	40,544.6-40,631.1
使用済み集合	Robin Hood による使用と検索	21.0	20.8-21.3
レートリミッター	GCRA、ハッシュ済みのクライアント	6.2	6.1-6.3
レートリミッター	GCRA、四つのルールスコープ、ハッシュ済みのクライアント	6.3	6.2-6.4
レートリミッター	GCRA、バイト列のクライアント（鍵付きハッシュを含む）	12.8	12.7-12.8

[23] benchmarks/results/README.md の「Method」と「Provenance and limits」（rustc 1.99.0。デーモンは cargo build --release -p sibuna でビルドした 49,113,856 バイト）、技術ガイド第 VIII 部「方法」。

サブシステム	ワークロード	中央値	最小-最大
レートリミッター	GCRA、四つのルールスコープ、バイト列のクライアント	13.1	13.0-13.2
HTTP パーサー	ゼロコピーのリクエストと Cookie	132.5	132.3-133.2
ポリシーエンジン	完全な分類、Shield のプロファイル	790.8	787.2-793.1
ポリシーエンジン	分類、Gate のプロファイル	75.9	75.6-76.1
WAF	8 KiB のボディー、意味解析による走査	12,784.3	12,771.5-12,809.2
WAF	8 KiB のボディー、SQL インジェクションを監査	15,485.4	15,460.9-15,530.1

ばらつきは小さく、どの行のどのバッチも、その行の中央値から 4.2% 以上離れていません[24]。デーモンはリクエストごとに一度だけクライアントをハッシュするので、ハッシュ済みのクライアントの行を使います。生のバイト列を渡す行は、検査のたびにその鍵付きハッシュを含みます。タイマーは個々の操作ではなくバッチ全体を囲むので、この表は操作ごとのパーセンタイルを報告しません。

8.3 製品全体

表 6 `wrk` による Sibuna の測定：毎秒リクエスト数、マイクロ秒単位のレイテンシー、マイクロ秒単位のリクエストあたり CPU 時間、最大常駐メモリ。3 回の実行の中央値（`benchmarks/results/product-latest.md`、2026-10-15T23:15:24Z 生成）

モード	運用構成	ワークロード（ステータス）	req/s	p50	p99	CPU	最大 KiB
forward auth	Gate	`admitted` (200)	211,345	151	305	11.84	15,912
forward auth	Gate	`challenged` (401)	198,315	160	326	13.15	16,176
forward auth	Gate	`allowed_static` (200)	211,053	152	305	11.73	16,176
forward auth	Gate	`attack` (200)	211,618	151	305	11.84	16,176
forward auth	Shield	`admitted` (200)	208,101	152	308	12.87	15,928
forward auth	Shield	`challenged` (401)	197,532	161	327	13.77	16,192
forward auth	Shield	`allowed_static` (200)	211,636	151	305	11.78	16,192
forward auth	Shield	`attack` (403)	210,549	152	307	12.32	16,192
リバースプロキシ	Gate	`admitted` (200)	142,498	427	768	27.68	15,896
リバースプロキシ	Gate	`challenged` (200)	199,465	160	324	12.98	16,160
リバースプロキシ	Gate	`allowed_static` (200)	147,303	411	759	26.88	16,324
リバースプロキシ	Gate	`attack` (200)	143,900	422	745	27.67	16,348
リバースプロキシ	Shield	`admitted` (200)	138,965	438	857	28.57	16,064
リバースプロキシ	Shield	`challenged` (200)	195,188	162	328	13.73	16,328
リバースプロキシ	Shield	`allowed_static` (200)	145,673	415	746	27.27	16,328
リバースプロキシ	Shield	`attack` (403)	209,507	151	306	12.17	16,328

forward-auth モードでは、イングレスがリクエストを許可するかどうかを Sibuna に問い合わせ、オリジンはありません。どのワークロードも毎秒 197,532 から 211,636 リクエストの範囲にあり、p99 は 305 から 327 us です。このときデーモンの四つの CPU は飽和しておらず（使用中は 2.47 から 2.72 個分）、上限を決めているのは負荷生成器だと考えられます。リバースプロキシモードでは、admitted と static の行にスタブのオリジンとループバックのホップ一つ分が加わり、デーモンの CPU は飽和しています（四つのうち 3.95 から 3.98 個分が使用中）。Sibuna が自ら応答するリクエスト（チャレンジと、Shield による攻撃の拒否）は、それでも毎秒約 195,000 から 210,000 件で動作します。最大常駐メモリは、どのワークロードでも 15,896 から 16,348 KiB の間にとどまります[25]。

リバースプロキシモードの三つの行で、約 200 万件の応答のうち数件が `503` でした。閉じかけの接続が、アドレスごとの接続制限の枠を返す前に、`wrk` が再接続したためです。現在のデーモンは、切断時の応答を書き込む前に枠を返します。これらのファイルは、その修正より前のものです。

[24] 技術ガイド第 VIII 部「プリミティブのレイテンシー」。`primitives-latest.json` から計算しています。

[25] `benchmarks/results/product-latest.md`。使用中の CPU 数は `product-latest.json` の `cores_busy_median` から。`503` の応答とその修正は `benchmarks/results/README.md` と SID 0002 の revision note（バグ修正）から。

表 7 Sibuna デーモンのフットプリント（product-latest.json と primitives-latest.json、2026 年 10 月 15 日、技術ガイド第 VIII 部の表による）

測定項目	値
デーモンのバイナリ、SQLite とコンソールを含む標準ビルド	49,113,856 バイト
アイドル時の常駐メモリ、製品の実行	5,592 から 5,788 KiB
ワークロード中の最大常駐メモリ	15,896 から 16,348 KiB
ワークロード後の常駐メモリ	16,176 から 16,348 KiB
ポリシーエンジンのテーブル	209,052 バイト
ボットシグネチャの表	42,244 バイト
WAF シグネチャの表	164,908 バイト
ブラウザーの証明器（WebAssembly）	14,744 バイト

その後のリリースプロファイルの変更で、同梱の SQLite のデバッグ情報が取り除かれました。ビルドマニフェストの注記によれば、49 MB のうち約 13 MB です。結果ファイルはそれ以降再生成していないので、測定したサイズを示しています[26]。

8.4 同じ負荷を与えたときのレイテンシー

wrk は常に 64 個のリクエストを処理中に保つので、リクエストのレートはデーモンに左右されます。また、wrk のパーセンタイルには、待っている間に送らなかったリクエストが含まれません。オープンループのハーネスは、代わりに与えるレートを固定し、すべての遅延をデーモンの負担として計上します。これが訪問者の体験に相当します。

forward-auth モードでは、毎秒 150,000 リクエストまで p99.9 が 75 us 未満にとどまります。オリジンが経路に入るリバースプロキシモードでは、毎秒 100,000 リクエストで p99 が 100 us、p99.9 が 150 us です。リバースプロキシモードの毎秒 150,000 リクエストでは、デーモンは処理能力を超えています。完了するのは毎秒 134,344 リクエストで、同じワークロードで wrk が測定した 138,965 に近く、レイテンシーは秒単位まで伸びます。処理能力を超えたときにオープンループのテストが示すべき振る舞いです。この行の四つのエラーは、過負荷の中で起きました[27]。

8.5 測定が変えたこと

レイテンシーのハーネスは、接続スレッドの設計を変えました。最初の実行では、64 個の接続をそれぞれ 4,096 リクエストの後にデーモンが閉じる設定で、毎秒 100,000 件の許可済みリクエストにおける 99.9 パーセンタイルは 0.3 から 2 ms でした。wrk では見えなかった裾です。スレッドの生成と終了では、シグナルスタックのマップとアンマップ、スタックのページの返却が起こり、こうしたアドレス空間の変更が他のスレッドを停滞させていました。一つの実験で診断が確定しました。接続を決して閉じず、したがって実行中にスレッドを生成しないビルドでは、同じレートで 54 から 60 us だったのです。修正では、接続の処理を終えたスレッドを待機させ、次の接続を引き受けさせます。10 秒間待機したスレッドは終了するので、バーストの後にスタックが常駐し続けることはありません。修正後、毎秒 100,000 リクエストでの p99.9 は、forward-auth モードで 62 us、リバースプロキシモードで 150 us です[28]。

表 8 Sibuna、Shield、一定の負荷レートでの許可済みリクエスト。マイクロ秒単位のレイテンシーとリクエストあたりの CPU 時間、10 秒間の実行 3 回の中央値（benchmarks/results/latency-latest.md、2026-10-15T23:07:14Z 生成）

モード	負荷 req/s	達成 req/s	p50	p90	p99	p99.9	エラー	CPU us
forward auth	10,000	10,000	30	32	37	58	0	16.92
forward auth	25,000	25,000	28	31	34	45	0	15.83
forward auth	50,000	50,000	25	27	30	40	0	13.83
forward auth	100,000	99,998	24	26	41	62	0	13.48
forward auth	150,000	149,996	25	36	51	74	0	14.16
リバースプロキシ	10,000	10,000	56	61	65	76	0	32.25
リバースプロキシ	25,000	25,000	51	54	58	72	0	28.63
リバースプロキシ	50,000	50,000	47	50	60	74	0	27.45
リバースプロキシ	100,000	99,996	55	75	100	150	0	28.42
リバースプロキシ	150,000	134,344	583,934	1,544,160	1,906,273	1,967,134	4	29.22

[26]技術ガイド第 VIII 部「フットプリント」、ルートの Cargo.toml のプロファイル[profile.release.package.libsqlite3-sys]。
[27]benchmarks/results/latency-latest.md。毎秒 25,000 リクエストの行と、毎秒 150,000 リクエストでのリバースプロキシの行は、このファイルにあります。
[28]SID 0002 の revision note（性能、接続スレッドの再利用）、技術ガイド第 VIII 部「測定が変えたこと」、benchmarks/results/latency-latest.md。

まれで周期的な停滞は、p99 を動かすよりずっと前に、分布の遠い裾を動かします。64 接続で毎秒 100,000 リクエストのとき、各接続は4096/1562.5 ≈ 2.6秒続くので、毎秒約 24 個のスレッドが生成され、終了します。100 万リクエストの実行では上位 0.1% はわずか 1,000 リクエストであり、スレッドの入れ替わりごとに数件の遅れたリクエストがあれば、それを埋めるのに十分です。このような欠陥を見るには、p99.9 と、すべてのリクエストに付けたタイマーが必要です。

8.6 要点

1. *1* マイクロ秒未満の分類。Gate のプロファイルでは、ポリシーエンジンはブラウザーのリクエストを 75.9 ns で分類します。Shield のプロファイルで完全な検査を行うと 790.8 ns です。
2. 対称鍵による検証。鍵付き BLAKE3 のセッショントークンの検証は 155.8 ns で、Ed25519 のトークンの 40,587.5 ns と比べて、一つの信頼ドメインの中で同じ完全性を約 260 分の 1 の作業量で得られます。
3. 小さく安定したフットプリント。四つの CPU で毎秒約 200,000 件の forward-auth リクエストを処理し、最大常駐メモリはどのワークロードでも 16 MiB 前後、アイドル時は 6 MiB 未満です。
4. 裾の測定にはオープンループが必要。スレッドの入れ替わりによる停滞は、クローズドループのハーネスでは見えず、オープンループの生成器の下では p99.9 に明らかに現れました。

9 市場での比較：Sibuna と近隣の製品

運用者が Sibuna と比較検討する可能性が最も高いツールと、Sibuna を比べます。他の製品についての事実は、2026 年 9 月に公開ドキュメントとリリース成果物から読み取ったもので、Sibuna 技術ガイドの第 II 部に出典とともに記録しています[31]。これらの製品はどれも利用者によく役立っており、それぞれ、私たちのハーネスでは測れない利点を持つ設計上の選択をしています。ここで述べるのはアーキテクチャの違いであり、順位ではありません。

9.1 Anubis との測定による比較

公式の Anubis v1.27.0 リリースバイナリを、製品全体のハーネスで Sibuna と並べて測定しました。四つのワークロードは同じで、各製品のスレッドにはすべて同じ四つの CPU を割り当て、`wrk` の負荷も同じです。Hashcash の作業量はそろえました（Sibuna はゼロビット 8 個、Anubis は 16 進数のゼロ 2 桁）。セッションは、ブラウザーと同じように各製品のチャレンジを解いて取得し、オリジンには `caddy respond` を使いました[29]。ここでは Sibuna は Shield の運用構成で動作するので、すべてのペイロードも検査します。Anubis はこれを行いません。

> **この表の読み方。** これらの数値は 1 台のマシン上の四つのワークロードを表すものであり、あらゆるデプロイを表すものではありません。また、いくつかの行は同じ条件での比較ではありません。forward-auth モードでは、Anubis は`/robots.txt` と攻撃のパスに自ら応答します。Anubis はペイロードを検査しないので、Shield による拒否のほうが多くの作業をしています。チャレンジの初期化は、Sibuna では二つのリクエスト、Anubis では一つのページです。セッション確認の差の多くは対称鍵のトークンで説明できます。Anubis 自身の HS512 方式は、標準の Ed25519 より 55% 速いからです。Anubis は広く使われている有能なツールであり、その保守者たちは正当な理由から異なる選択をしています。この測定が示しているのは、小さく有界なリクエスト経路によって、防御側にとって許可を安価にできるということです。

forward-auth モードでは、Anubis は許可したリクエストあたり 106 us の CPU 時間を使い、Sibuna は 13 us です。Anubis の最大常駐メモリは forward-auth モードで約 32 MiB、リバースプロキシモードでは約 600 MiB まで増えましたが、Sibuna は終始 16 MiB 前後にとどまりました。`caddy respond` の後ろでは、Sibuna のプロキシした行はオリジンによって制限されます。オリジンが負荷生成器と CPU を共有しているからです。製品全体のより軽いスタブのオリジンを使うと、同じ行は毎秒約 139,000 リクエストに達します。

9.2 アーキテクチャの詳細な比較

9.2.1 SafeLine（Chaitin Technology）

SafeLine は、libinjection の意味解析の考え方を、管理コンソールを備えた完全な製品にしたものです。ベンダーは、検出率 71.65%（バランスモード）と 76.17%（厳格モード）、誤検知率 0.07% と 0.22% を報告しています[32]。これらは検出品質の測定値であり、Sibuna は自身のエンジンについてこの種の値を公表していません。

- デプロイ。コミュニティ版は、少なくとも 1 CPU コア、1 GB のメモリ、5 GB のディスクを持つ Linux ホスト上の七つの Docker コンテナーとして提供されます。TLS は自ら終端しますが、Sibuna はそうしません。
- 状態。攻撃ログはスタック内の PostgreSQL に保存し、スタックはホストごとに一つです。
- *Sibuna* との違い。Sibuna は任意の SQLite ファイルとともに起動する一つのバイナリであり、クラスターでは運用者がすでに運用しているかもしれない一つの PostgreSQL データベースを共有します。製品の実行におけるア

[31] 技術ガイド第 II 部「機能比較」と「中間ページとエッジ」。Sibuna のライセンスは `Cargo.toml` にあるワークスペースの `license` です。
[29] `benchmarks/results/comparison-latest.md` と `admission-latest.md`。2026 年 10 月 16 日に `cargo xtask bench product --anubis` と `cargo xtask bench admission --anubis` で記録しました。技術ガイド第 VIII 部「Sibuna と Anubis」に、方法と製品間のすべての違いを挙げています。
[32] 技術ガイド第 VIII 部「測定していないもの」：ベンダーが報告した数値を 2026 年 9 月に確認しました。

表 9 機能の比較（技術ガイド第 II 部。他の製品についての事実は 2026 年 9 月に公開ドキュメントから読み取ったもの、JA4 の行は 2026 年 10 月）

機能	Sibuna 1.0.0-beta.3	Anubis 1.27	SafeLine CE 9	Cloudflare WAF
実行形態	一つのバイナリ。リバースプロキシまたは forward auth。任意で SQLite ファイルまたは共有 PostgreSQL データベース	一つの Goバイナリ。リバースプロキシまたは forward auth	七つの Docker コンテナー（tengine、detector、mgt、luigi、fvm、chaos、PostgreSQL）	ホスト型のネットワーク。DNS を Cloudflare に向ける
ランタイム	ネイティブコード、ガベージコレクターなし。ストレージを無効にすると第三者の依存関係なし	Go、ガベージコレクションあり	C/C++のプロキシと検出器、Go の管理機能、PostgreSQL	独自のエッジソフトウェア
ライセンス	LGPL-3.0-or-later	MIT	管理コードは GPL-3.0。検出器のイメージは非公開	独自のサービス
作業証明による許可	Hashcash（ビット単位）と Cohen-Pietrzak の逐次作業。ルールごとに選択	16 進数の桁単位の SHA-256 Hashcash。作業を伴わないチャレンジもあり	JavaScript チャレンジと CAPTCHA。作業に基づかない	マネージドチャレンジ、JS チャレンジ、Turnstile
サーバー上のチャレンジの状態	解かれるまでなし。解かれたタグは固定サイズの Robin Hood テーブルに保存	ストアのバックエンド：メモリ、bbolt、Valkey、S3	スタックが管理	Cloudflare が管理
セッショントークン	鍵付き BLAKE3、75 文字、支払った作業に結び付く。Ed25519 も選択可	Ed25519 で署名した JSON Web Token。HS512 も選択可	スタックが発行する Cookie	`cf_clearance` Cookie
ペイロードの検査	Shield：タグ付きのオートマトンと構造トークナイザー、ボディーの先頭 8 KiB	なし	OWASP のカテゴリーに対する意味解析	マネージドルールセット（Pro 以上）
TLSフィンガープリント（JA4）	信頼できるイングレスが転送する JA4 値とルールで照合。Sibuna は計算しない	文書化されていない。ポリシーは Anubis が計算する HTTP フィンガープリント JA4H を参照可能	自身の TLS 終端で計算。カスタムルールと組み込みの拒否リスト	カスタムルールと分析で JA3 と JA4（Bot Management 付きの Enterprise）
共有状態	ポリシー、レピュテーション、フォレンジックのための一つの PostgreSQL データベース	共有の署名鍵。共有の Valkey ストア	ホストごとに一つのスタック	グローバルなエニーキャストネットワーク
TLS 終端	なし（イングレス）	なし（イングレス）	あり	あり

表 10 `wrk` による Sibuna（Shield）と Anubis 1.27.0 の測定：毎秒リクエスト数とマイクロ秒単位の p99 レイテンシー、5 秒間の実行 3 回の中央値（`benchmarks/results/comparison-latest.md`）。示していない場合のステータスは 200 です。Shield は攻撃を 403 で拒否します。

モード	ワークロード	Sibuna req/s	Sibuna p99	Anubis req/s	Anubis p99
forward auth	`admitted`	208,635	308	36,509	7,477
forward auth	`challenged`	193,087	342	113,170	3,781
forward auth	`allowed_static`	212,444	304	89,904	4,547
forward auth	`attack`	211,347 (403)	306	36,853	7,283
リバースプロキシ	`admitted`	103,298	3,321	19,398	11,062
リバースプロキシ	`challenged`	197,327	328	28,842	9,546
リバースプロキシ	`allowed_static`	106,819	3,316	28,862	30,124
リバースプロキシ	`attack`	210,483 (403)	306	23,399	14,124

表 11 2 つの CPU での毎秒の許可処理の操作数、七つのバッチの中央値（`benchmarks/results/admission-latest.md`）。Anubis は、トークン方式ごとに 1 回ずつ実行しました。

製品（トークン）	セッションの確認	未認証	初期化	証明の検証	常駐メモリ
Sibuna（鍵付き BLAKE3）	88,104	78,997	42,664	57,924	7.6 MiB
Anubis（Ed25519）	10,951	28,619	7,893	12,668	32.6 MiB
Anubis（HS512）	16,976	27,785	7,952	19,261	29.1 MiB

イドル時の常駐メモリは 5,592 から 5,788 KiB でした。SafeLine は、Docker デーモンのない私たちのホストでは動かしていないので、性能の比較はしていません。

9.2.2 Anubis と Coraza

作業証明の中間ページが一般的になったのは 2025 年のことです。言語モデルの学習のためのスクレイパーのトラフィックによって、普通のオリジンの運用コストがかさむようになりました。Anubis はその最もよく知られたオープンな実装です。これは広くデプロイされている有能な Go のリバースプロキシで、JavaScript のパズルを配信し、JSON Web Token に Ed25519 で署名し、チャレンジの状態をストアに保持します。Coraza は ModSecurity のルール言語（SecLang）を実装し、OWASP Core Rule Set を実行します。このようなゲートと並べて、ルールベースの検査を加える方法の一つです。

- 作業はどこに行くか。Anubis も Sibuna Gate も、オリジンが何かをする前にクライアントに代価を支払わせます。Sibuna はさらに、より強いゲートに値するルートのために、ルールごとに選べる逐次作業の第二層を加え、そのセッションはどれだけの作業が支払われたかを記憶します。
- サーバーが記憶するもの。Anubis はチャレンジを保存します。Sibuna は自己認証型のレコードを発行し、解かれたタグだけを固定サイズのテーブルに記憶します（定理 3）。標準のトークンは対称鍵であり、検証が安価で、量子的な攻撃者にも二次的な高速化しか許しません。
- 検査。Sibuna Shield は許可と同じ走査の中で独自の構造的な検査を行うので、一つのプロセスと一つのパーサーが両方の役割を担います。OWASP Core Rule Set は実行しません。libmodsecurity と Coraza へのバインディングを含め、既存の SecLang エンジンを評価しましたが、どれも採用しませんでした。エッジサーバー上の検査エンジンはセキュリティ上重要なコードであり、自分たちで所有しレビューしたいからです。Sibuna が Core Rule Set との互換性を加える場合は、まず設計記録でそのサブセットを規定します[33]。
- 測定。同じ負荷の下で 1 台のホストにおいて、Sibuna は forward-auth モードで 5 倍以上の許可済みリクエストに応答し、p99 はより低く、リクエストあたりの CPU 時間は約 8 分の 1 でした（Anubis との測定による比較）。

9.2.3 クラウドのエッジ WAF（Cloudflare など）

ホスト型のエッジは、検査、レート制限、ボットのスコアリングを多数の拠点（PoP）に置き、グローバルなコントロールプレーンで管理します。単一のサーバーでは吸収できない大規模な攻撃を吸収し、運用者のハードウェアも必要としません。Sibuna のクラスター構成は、共有のポリシープレーンがローカルの不変なスナップショットに供給するという形をそれらから借りつつ、セルフホストのままです。

- データの所在。ホスト型の WAF は運用者の TLS を終端し、共有のインフラストラクチャ上でリクエストのペイロードを見るので、データの所在に関する要件と衝突することがあります。
- プランと上限。機能はプランに結び付いています。Cloudflare が公開している上限では、Free、Pro、Business、Enterprise の各プランで、カスタムルールは 5、20、100、1,000 個、レート制限ルールは 1、2、5、100 個です。
- *Sibuna* との違い。Sibuna は運用者自身のマシンで動作し、リクエストごとの料金はかかりません。ポリシーの変更は、次のストレージのティック（標準は 500 ms）ですべてのノードに届きます。大規模なネットワーク攻撃の緩和は行わず、学習済みモデルによるボットのスコアリングも行わず、TLS も終端しません。それらを必要とするデプロイでは、引き続き前段にイングレスやエッジネットワークが必要です。

10 実装：小さなクレートと少ない依存関係

Sibuna は Rust で書かれています。小さなクレートからなるワークスペースで、依存関係は一方向を向いています。再利用可能なライブラリ、次に設定と診断の中核、次にアプリケーションのライブラリ、最後にデーモンとブラウザーの証明器です。運用者にとって重要な工学上の選択が三つあるので、簡潔に記録しておきます[34]。

少ない依存関係。第三者のライブラリはどれも負債とみなします。レビューし、更新し、信頼しなければならないからです。データプレーンは一つも使いません。ハッシュ、署名、base64、JSON、DEFLATE、Aho-Corasick、HTTP/1.1 パーサーは私たち自身の小さなライブラリで、他のプロジェクトも単独で利用できます。依存関係を採用するのは、それが成熟し、自己完結し、本当に大きな問題を解決する場合だけで、その理由はビルドマニフェストの該当箇所と設計記録に書きます。採用したのは二つで、それぞれ一つのライブラリに閉じ込めてあります。SQLite と PostgreSQL のストレージのための `sqlx`（推移的に約 100 個のクレートを伴います[35]）と、コンソールの外向きの TLS と PostgreSQL の TLS のための `rustls`（実行時に 13 個のクレート）です。後者を採用したのは、自前で書く TLS クライアントは、Sibuna のデータプレーンより大きな、セキュリティ上重要なプロジェクトになるからです[36]。ストレージとコンソールなしのビルドには、第三者の依存関係がまったくありません。

メモリ安全性。コンパイラーのメモリ安全性の検査を迂回するコードは標準で禁止しており、それを必要とするモジュールは名前を挙げて明示的に許可し、使う箇所ごとにコメントで理由を説明しなければなりません。いくつかのモジュールがそうしています。プロセッサーの SHA 命令を使う経路、秘密情報を消去する一つの書き込み、シグナ

[33] SID 0010「The OWASP Core Rule Set」。
[34] SID 0010「Workspace」「Dependency policy」「Engineering standards」。
[35] SID 0011「Why sqlx」、`Cargo.toml`（第三者のクレートにはそれぞれ採用の理由を記しています）。
[36] SID 0012「Specification」「Rationale and alternatives」と review history（PostgreSQL の TLS のビルド）。

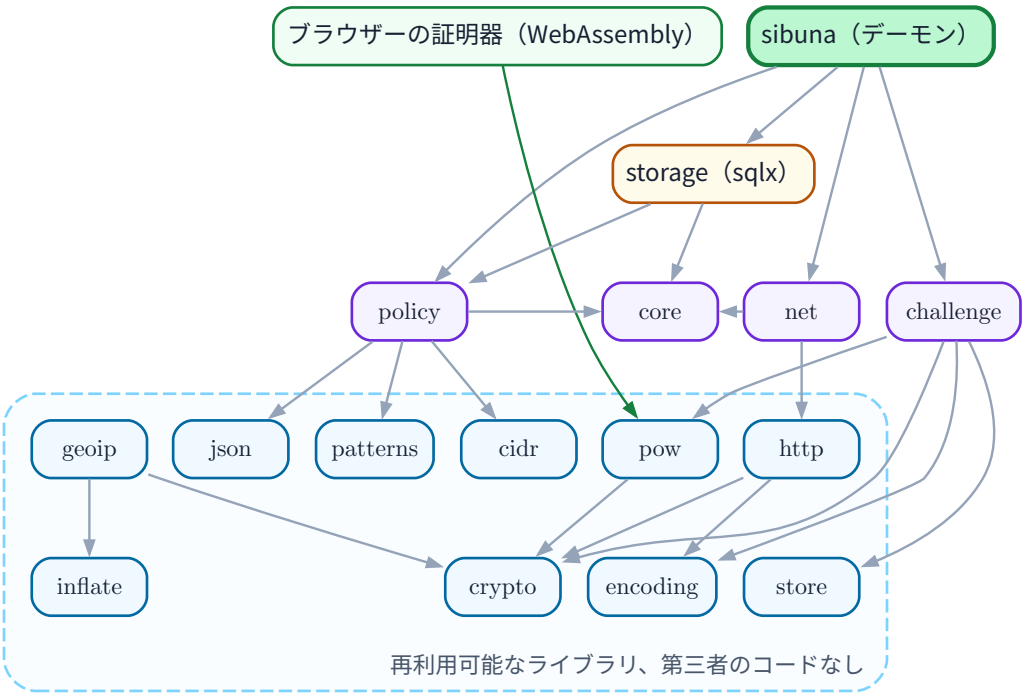


図 6 Sibuna のライブラリ（簡略化、SID 0010 による）。矢印は、ライブラリからそれが使うものへ向かいます。再利用可能なライブラリはアプリケーションのライブラリに依存せず、第三者のコードを持ち込むのはストレージとコンソールのライブラリだけです。

ル処理のためのいくつかの C ライブラリ関数の宣言、二つの WebAssembly モジュールのエクスポート、そしてインシデントをストレージスレッドに運ぶリングです。速度のためにこのようなコードを残したのはリングだけで、それも測定の後のことです。標準ライブラリの有界チャネルは、16 個の生産者で一回のプッシュあたり 20 から 45% 多くのコストがかかりました[37]。それ以外の箇所では、手書きのメモリ管理が担うはずの仕事を所有権が担います。解析済みのリクエストはバッファーより長く生存できず、エンジンは最後の読み手とともに解放され、ソケットはどのスレッドからもまだ到達できる間は閉じられません。

規律とテスト。関数の本体は最大 70 論理行、ファイルは最大 1,408 行で、チェッカーが強制します。ライブラリのコードは構造化されたエラーを返し、決して出力しません。また、利用者が目にするエラーはすべて、その位置を示し、抜粋を表示し、ヒントを与えます。1.0.0-beta.3 のワークスペースのスイートは、SQLite で約 2,200 個、PostgreSQL に対してさらに約 700 個のテストを実行します。コンソールのインターフェースにはゴールデンフィクスチャを含む 298 個のテストがあり、ブラウザーの検査はヘッドレス Chromium で中間ページとコンソールのすべてのページを操作します[30]。

11 限界と今後の課題

どの主張もその証拠より広く読まれることのないよう、本稿が立証していないことを挙げます。

- 測定した近隣の製品は一つ。Anubis は 1 台のホストで測定しました（Anubis との測定による比較）。SafeLine もホスト型の WAF も測定していません。
- 負荷をかけたクラスターはなし。機能テストでは一つの PostgreSQL データベース上で三つのノードを動かしています（SID 0011、Committed）が、複数のノードに負荷をかけるハーネスはありません。
- *PostgreSQL* にはソースからのビルドが必要。リリース版の実行ファイルは SQLite のストレージとコンソールを含みますが、PostgreSQL のサポートは含みません。クラスターでは `postgres-tls` フィーチャー付きで Sibuna をビルドします。
- コンソールの分離の判定、すなわち負荷の下で八つのダッシュボードを動かす判定は、まだ実施していません（コンソール）。
- ループバックのみ。TLS、広域ネットワーク、負荷の下での本物のブラウザーを対象とした測定はありません。デプロイでは、それぞれの経路を自ら測定すべきです。

[37]SID 0010 の review history（セキュリティレビュー、2026-10-15）：標準の有界チャネルと Vyukov のリングの比較、`tests/timing.rs`。明示的に許可したモジュールは `crates/crypto/src/sha256/`（x86-64 と AArch64）、`crates/crypto/src/ct.rs`、`cmd/sibuna/src/signals.rs`、`cmd/sibuna-pow-wasm/src/lib.rs`、`cmd/console-ui/src/abi.rs`、`crates/store/src/ring.rs` です。ワークスペースは `unsafe_code = "deny"` と Clippy の `undocumented_unsafe_blocks = "deny"` を設定しています。

[30]SID 0010 の「Acceptance criteria and implementation status」（criteria 1、3、5）。SID 0014 より前の時点で、ワークスペースのテスト約 2,100 個とインターフェースのテスト 290 個を記録しています。1.0.0-beta.3 の件数は `cargo test --workspace -- --list` によるものです。

- 有界な検査。Shield はボディーの先頭 8 KiB を対象とするヒューリスティックの検査器であり、完全な SQL パーサーや HTML パーサーではありません。また、forward-auth での検査が見るのは、イングレスが送るバイトだけです。
- プロトコルの範囲。Sibuna は HTTP/1.0 と HTTP/1.1 を扱い、HTTP/2 にも受信側の TLS 終端にも対応していません。
- *TLS* フィンガープリントはイングレスから得る。Sibuna は JA4 を自ら計算できません。フィンガープリントの信頼性は、それを書き込むイングレスの信頼性を超えず、そのイングレスはクライアントが送ったどの写しも上書きしなければなりません。クライアントは専用のライブラリでブラウザーのハンドシェイクを模倣できるので、JA4 は自動化のコストを引き上げますが、身元を証明するものではありません[11]。
- 暗号学的な保証。逐次作業の層は、ランダムオラクルモデルにおける公表済みの証明に基づいています。私たちの具体的なプロトコルには参照用のラベル計算器に対するテストがありますが、独立した暗号監査は受けていません。
- *1* 台のマシン。すべての数値は 1 台の AMD Ryzen 9 5950X で得たものです。このホストでの順位は、別のホストでの順位ではありません。

12 結論

Web 防御は、しばしば継ぎ足しによって大きくなってきました。インタープリターのランタイム、コンテナー、正規表現のルールセット、外部ストアが、アプリケーションの前に積み上げられてきたのです。Sibuna では、いくつかの素朴な考え方に導かれて、その逆の方向を試みました。

- 送信者に検証可能な作業で許可の代価を支払わせ、サーバーの検査は安価で状態を持たないものに保ち、セッションにはどれだけの作業を支払ったかを記憶させる。
- 許可と同じ走査の中で、構造に基づいて線形時間で検査し、どのセッションも検査を回避できないようにする。
- マシンを尊重する。接続ごとに一つのバッファー、リクエスト経路での割り当てなし、鍵付きの配置による有界なテーブル、作り直すのではなく再利用するスレッド。そして、
- 永続的な共有状態は成熟したデータベースに任せ、リクエスト経路が決して待つことのない一つのストレージスレッドの背後に置く。

16 コアのマシン 1 台で、これらの選択により、Sibuna は四つの CPU を使って forward-auth モードで毎秒 200,000 件を超える許可済みリクエストに応答し、その負荷の下での常駐メモリは約 16 MiB、毎秒 100,000 リクエストでの p99.9 は 62 us でした。これらは 1 台のホストでループバックを使って得た数値であり、順位としてではなく、設計がまとまって機能していることの証拠として示します。私たちは独自の合意プロトコルを取り除き、そのことを率直に述べます。その代わり、システムはより小さくなり、データプレーンには第三者の依存関係がありません。測定すべきことはまだ多く残っており、それが何かを正確に述べるよう努めました。

13 出典と再現

本稿のすべての数値は、Sibuna リポジトリにコミットされた結果ファイル、Sibuna Discussion（SID）の記録、または Sibuna 技術ガイドに由来し、各表や脚注がその出典を示しています。結果ファイルは `cargo xtask bench primitives`、`cargo xtask bench product`、`cargo xtask bench latency` が書き出し、`benchmarks/results/README.md` に各ファイルのコマンド、CPU セット、来歴を記しています。

- `benchmarks/results/primitives-latest.md`、`product-latest.md`、`latency-latest.md` と、それぞれの `.json` ファイル。`console-8-dashboards-2026-10-15.md`。
- `docs/sid/records/` にある SID 0002（基礎アーキテクチャとその改訂注記）、SID 0004（Shield の運用構成）、SID 0006（数学的基礎）、SID 0007（コンソール）、SID 0010（ワークスペース、クレート、依存関係の方針）、SID 0011（sqlx によるストレージ）、SID 0012（コンソールの TLS）、SID 0013（コンソールのストレージ）、SID 0014（イングレスから受け取る JA4 フィンガープリント）。
- Sibuna 技術ガイド、`docs/book/`：第 I 部（リクエストのコスト）、第 II 部（先行事例）、第 III 部（暗号）、第 IV 部（プロトコル）、第 V 部（ゼロアロケーションのエンジニアリング）、第 VI 部（オートマトンと意味解析ファイアウォール）、第 VIII 部（実験による評価）、第 IX 部（運用）。
- 一次文献：Back, Hashcash (1997)、Cohen and Pietrzak, Simple Proofs of Sequential Work (EUROCRYPT 2018)、Blocki, Lee and Zhou, On the Security of Proofs of Sequential Work in a Post-Quantum World (ITC 2021)、Aho and Corasick, Efficient String Matching (1975)、Celis, Robin Hood Hashing (1986)、ATM Forum の Generic Cell Rate Algorithm (1996)。