SIBUNA

शून्य-आवंटन वाला, कार्य-आधारित वेब सुरक्षा इंजन

कार्य-आधारित प्रवेश, रैखिक-समय निरीक्षण, सीमित अवस्था और SQL के ज़रिए स्टोरेज

Vikrant Rathore

Ronak Rathore और Kanak Rathore के सहयोग से
Insan AI | रिलीज़ 1.0.0-beta.3

सार

वेब एप्लिकेशन फ़ायरवॉल और edge सुरक्षा प्रणालियाँ प्रायः तीन आर्किटेक्चरल समस्याओं से ग्रस्त होती हैं। पहली है लागत का उलटाव: भेजने वाला एक अनुरोध भेजने में लगभग कुछ खर्च नहीं करता, जबकि रक्षक उसे पार्स करता है, बैकट्रैक कर सकने वाले रेगुलर एक्सप्रेशन से स्कैन करता है, और उसके बारे में किसी बाहरी स्टोर से पूछता है। दूसरी है रनटाइम की अप्रत्याशितता: अनुरोध पाथ पर हीप आवंटन, garbage collection के ठहराव, और कई कंटेनर वाले डिप्लॉयमेंट लेटेंसी वितरण की टेल को सीमित करना कठिन बना देते हैं। तीसरी है संचालन का बोझ: प्रतिबंधों, टोकन और सीमाओं की साझा अवस्था अक्सर ऐसी सेवाओं में रहती है जिन्हें ऑपरेटर को फ़ायरवॉल के साथ चलाना पड़ता है। **Sibuna** एक ही स्व-निहित बाइनरी में थोड़े-से सरल तंत्रों के साथ इन समस्याओं को हल करने का हमारा प्रयास है, और यह पत्र हर तंत्र के पीछे का तर्क समझाता है। (i) कार्य-आधारित प्रवेश: ब्राउज़र प्रति नियम चुनी गई Hashcash पहेली या Cohen-Pietrzak अनुक्रमिक कार्य का प्रमाण हल करके सत्र अर्जित करता है, और सर्वर उत्तर का सत्यापन ऐसे चैलेंज के विरुद्ध 70.2 ns या 22.3 us में करता है जिसे उसने कभी संग्रहीत नहीं किया। (ii) सीमित, आवंटन-मुक्त अनुरोध पाथ: byte-class Aho-Corasick ऑटोमेटन 40 बॉट सिग्नेचर का मिलान 55.7 ns में करता है, जो अनुक्रमिक खोज से 13.7 गुना तेज़ है; GCRA लिमिटर प्रति क्लाइंट 16 बाइट की अवस्था के साथ 6.2 ns में निर्णय लेता है; और Robin Hood spent set किसी समाधान को 21.0 ns में दर्ज करता है। SQL injection, cross-site scripting, command injection और path traversal का निरीक्षण उसी पास में, इनपुट में रैखिक समय में चलता है। (iii) अपने किसी *consensus* प्रोटोकॉल के बिना साझा अवस्था: एक नोड के लिए एम्बेडेड SQLite फ़ाइल और क्लस्टर के लिए एक साझा PostgreSQL डेटाबेस, जिसे एक स्टोरेज थ्रेड पढ़ता है और जिसकी प्रतीक्षा अनुरोध कभी नहीं करते। एक AMD Ryzen 9 5950X पर, चार CPU पर चलते डेमन के साथ, Sibuna प्रति सेकंड लगभग 200,000 forward-auth अनुरोधों का उत्तर देता है (वर्कलोड के अनुसार 197,532 से 211,636), 305 से 327 us के p99 के साथ; प्रति सेकंड 100,000 अनुरोधों के open-loop लोड में उसका p99.9 62 us रहता है; और उसकी resident मेमोरी का पीक लगभग 16 MiB है। हम बताते हैं कि हर संख्या कैसे प्राप्त हुई, और उन चीज़ों की सूची देते हैं जिन्हें हमने नहीं मापा।

6.2 ns	55.7 ns	22.3 µs	16 MiB
GCRA रेट जाँच	**बॉट मैचर**	**PoSW सत्यापन**	**डेमन का पीक RSS**
हैश किया गया क्लाइंट | 16 शार्ड	Aho-Corasick | 40 सिग्नेचर	गहराई 13 | 16 ऑपनिंग	wrk लोड में | 4 CPU

एक AMD Ryzen 9 5950X कोर पर Sibuna, डेमन के लिए चार CPU, 15 अक्टूबर 2026: benchmarks/results/primitives-latest.md और product-latest.md; विधि मूल्यांकन अनुभाग में है।

1 प्रस्तावना: एक अनुरोध का अर्थशास्त्र

सर्वर तक पहुँचने वाला हर अनुरोध उसके ध्यान पर एक दावा करता है। एक मेथड, एक पाथ और कुछ हेडर यह स्थापित नहीं करते कि अनुरोध को सर्व करना सार्थक है। इसलिए कोई सुरक्षा प्रणाली अनुरोध और ऑरिजिन के बीच एक निर्णय रखती है: उसे प्रवेश दें, अस्वीकार करें, या भेजने वाले से पहले जाँचा जा सकने वाला कार्य करने को कहें। यह पत्र Sibuna में उस निर्णय को उसके गणित से लेकर मेमोरी में रखे बाइट तक देखता है।

> **Feynman, भौतिक अंतर्ज्ञान पर: कार्य की कीमत कौन चुकाता है?**
>
> एक ऐसे टर्नस्टाइल की कल्पना करें जो हल्के-से स्पर्श पर घूम जाता है। उसके पीछे एक पहरेदार बैठा है, जिसे हर स्पर्श पर खड़ा होना, तीन बहीखाते जाँचने और मुख्यालय को फ़ोन करना पड़ता है। बाहर खड़ा कोई बच्चा पूरी दोपहर बिना मेहनत टर्नस्टाइल छूता रह सकता है, जबकि पहरेदार दोपहर के भोजन से पहले ही थक जाता है। कई फ़ायरवॉल की व्यवस्था यही है: सस्ता पक्ष हमलावर का है। भौतिकी एक बेहतर व्यवस्था सुझाती है। टर्नस्टाइल को एक पंप से जोड़ दें, ताकि जो भी पार जाना चाहे उसे गेट के ऊपर रखी टंकी में एक बाल्टी पानी चढ़ाना पड़े। पानी चढ़ाने में थोड़ी ईमानदार मेहनत लगती है; पहरेदार बस टंकी पर नज़र डालकर देख लेता है कि पानी ऊपर गया। नज़र डालने में पहरेदार का लगभग कुछ नहीं लगता, और पानी चढ़ाने में आगंतुक का कुछ वास्तविक लगता है। रोज़ आने वाला यात्री एक बार पानी चढ़ाता है और पूरे दिन आता-जाता है; शरारती जल्द ही घर लौट जाता है।

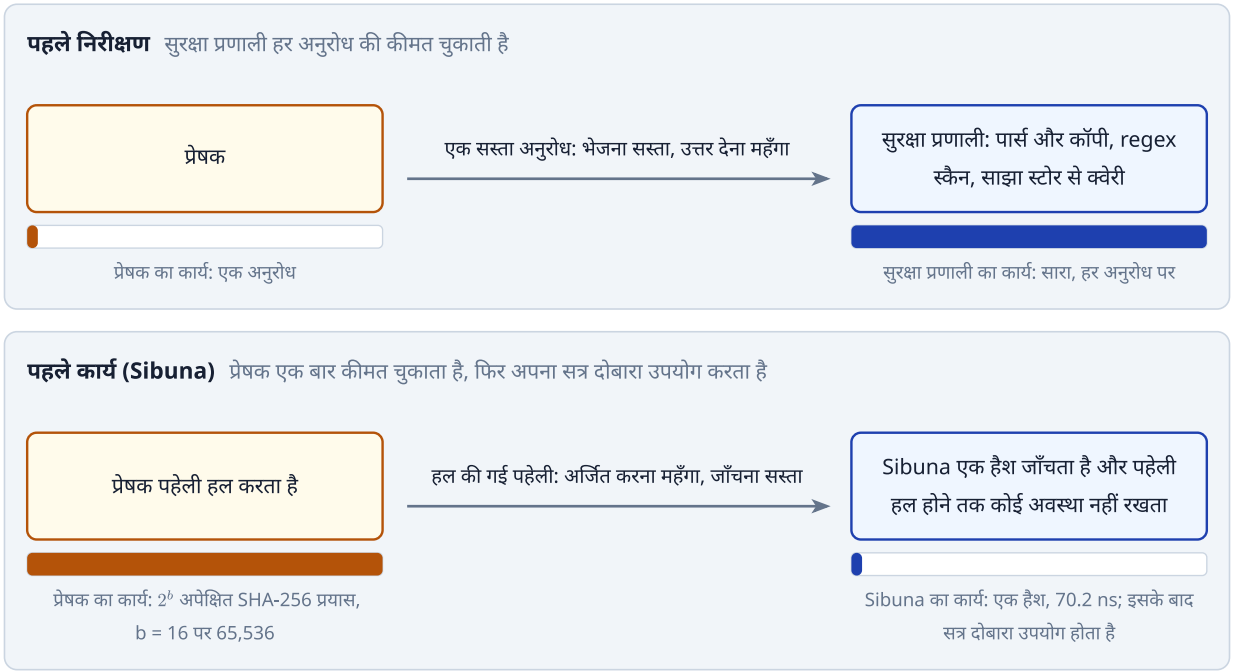
इस पत्र के voice बॉक्स हमारे अपने शब्द हैं, जो Richard Feynman, Donald Knuth और Leslie Lamport की भावना में लिखे गए हैं, जिनकी समझाने की शैली का हम अनुसरण करने का प्रयास करते हैं। ये उद्धरण नहीं हैं, सिवाय वहाँ के जहाँ कोई बॉक्स अपने स्रोत का नाम देता है।

एक स्वचालित प्रेषक अपने भेजे हर अनुरोध के लिए बहुत कम चुकाता है। जब कोई अनुरोध किसी पारंपरिक फ़ायरवॉल तक पहुँचता है, तो रक्षक को TLS handshake पूरा करना, अनुरोध को कॉपी करने और बाँटने के लिए मेमोरी आवंटित करना, उस पर रेगुलर एक्सप्रेशन की एक सूची चलाना, और किसी काउंटर को अपडेट करने के लिए साझा स्टोर तक नेटवर्क round trip करना पड़ सकता है। हम इस असमानता को कोई सार्वभौमिक अनुपात नहीं देते, क्योंकि ऐसा कोई अनुपात है ही नहीं: यह ऑरिजिन, cache और आक्रमण पर निर्भर करता है। इसके बजाय एक सरल मॉडल निर्णय को दृश्य बनाता है। मान लें c_o प्रति स्वीकृत अनुरोध ऑरिजिन का कार्य है, c_g प्रति अनुरोध गेट का कार्य, और a उन अनुरोधों का अंश जिन्हें गेट प्रवेश देता है। गेट अपनी लागत तभी वसूल करता है जब

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

यानी, जब वह ऑरिजिन का जितना कार्य बचाता है वह उसकी अपनी लागत से अधिक हो[1]। इससे दो परिणाम निकलते हैं। गेट की अपनी लागत c_g इनपुट चाहे जो हो, छोटी और सीमित होनी चाहिए, और प्रवेश का निर्णय सर्वर के लिए नहीं, भेजने वाले के लिए महँगा होना चाहिए।

[1] पुस्तक का भाग I, "प्रवेश एक आर्थिक निर्णय है"। यह असमिका क्षमता का मॉडल है, मापन नहीं; इसमें बैंडविड्थ, कनेक्शन की अवस्था और क्लाइंट की देरी शामिल नहीं हैं।



चित्र 1 कार्य कहाँ जाता है। ऊपर, पहले निरीक्षण करने वाली सुरक्षा प्रणाली प्रेषक के भेजे हर अनुरोध पर अपना कार्य खर्च करती है। नीचे, Sibuna प्रेषक से 2^b अपेक्षित हैश प्रयास माँगता है और उत्तर का सत्यापन एक हैश से करता है। मीटर एक मॉडल हैं; सत्यापन का समय मापा गया है (16 बिट पर Hashcash सत्यापन, `benchmarks/results/primitives-latest.md`)।

Sibuna प्रवेश को कार्य के प्रमाण पर निर्भर बनाकर इस असमानता को उलट देता है। सत्र अर्जित करने के लिए ब्राउज़र ऐसा नॉनस खोजता है जिसके SHA-256 डाइजेस्ट में b शुरुआती शून्य बिट हों, या अनुक्रमिक कार्य का प्रमाण गणना करता है। अपेक्षित खोज 2^b हैश प्रयास है, $b = 16$ पर 65,536; सर्वर एक उम्मीदवार जाँचता है। सर्वर की जाँच स्टेटलेस भी है: चैलेंज जारी करने में कुछ नहीं लिखा जाता, और केवल चुकाया गया समाधान ही मेमोरी घेरता है।

कार्य का प्रमाण यह स्थापित नहीं करता कि भेजने वाला मनुष्य है, न उसकी पहचान, न उसका इरादा। प्रेषक कंप्यूटर किराये पर ले सकता है, किसी सत्र को उसकी अवधि के भीतर साझा कर सकता है, या कार्य को कई मशीनों में बाँट सकता है। पहेली प्रवेश की कीमत बदलती है; स्वीकृत अनुरोध क्या कर सकता है, यह अब भी नीति और निरीक्षण तय करते हैं।

2 डिज़ाइन का अवलोकन: एक पास में तीन निर्णय

Sibuna प्रवेश, निरीक्षण और प्रतिष्ठा को तीन अलग-अलग निर्णय रखता है, और ऑरिजिन के कोई भी कार्य करने से पहले इन्हें एक ही प्रोसेस में, अनुरोध पर एक ही पास में लेता है। वह इन्हें उत्पाद की दो सतहों के रूप में देता है। *Gate* नियमों, CIDR प्रतिष्ठा और बाढ़ पर स्थानीय नियंत्रण के साथ proof-of-work प्रवेश देता है। *Shield*, जो डिफ़ॉल्ट है, अनुरोध पेलोड का सीमित निरीक्षण जोड़ता है। दोनों में से कोई भी सतह ऑरिजिन के सामने रिवर्स प्रॉक्सी के रूप में चलती है, या forward-auth endpoint के रूप में, जिससे Caddy या Nginx जैसा कोई इनग्रेस पूछता है। नीति, प्रतिष्ठा और घटना इतिहास का स्थायी स्टोरेज दोनों के लिए एक विकल्प है: एक नोड पर स्थानीय, या किसी क्लस्टर द्वारा साझा[2]।

तालिका 1 उत्पाद की दो सतहें (SID 0002; पुस्तक का भाग II)

क्षमता	Gate (`--gate`)	Shield (`--shield`, डिफ़ॉल्ट)
प्रवेश (Admission)	Hashcash या अनुक्रमिक कार्य वाला ब्राउज़र चैलेंज; क्लाइंट से और उसके चुकाए कार्य से बँधा सत्र टोकन	वही
नीति	घोषणात्मक नियम (पाथ, User-Agent, हेडर, IPv4 और IPv6 CIDR, विश्वसनीय इनग्रेस से JA4 TLS फ़िंगरप्रिंट), ALLOW, DENY, CHALLENGE और WEIGH के साथ; बॉट सिग्नेचर; प्रतिष्ठा	वही
बाढ़ नियंत्रण	GCRA रेट लिमिट, हनीपॉट, अस्थायी प्रतिबंध	वही
पेलोड निरीक्षण	अक्षम	बॉडी के पहले 8 KiB पर SQL injection, cross-site scripting, command injection और path traversal
वैकल्पिक अवस्था	SQLite में, या PostgreSQL के ज़रिए साझा, नीति और प्रतिष्ठा	साथ में घटनाओं की फ़ॉरेंसिक जाँच: full-text खोज और कैंपेन क्लस्टरिंग

[2]SID 0002, "Decision"; पुस्तक का भाग II, "एक बाइनरी में तीन विचार"; पुस्तक का भाग VI, "घोषणात्मक नीति इंजन"। प्रति-नियम उदाहरण, आवश्यकता टिकट और वर्क-लेवल वाला सत्र पुस्तक के भाग IV से हैं; अनुकूली आधार रेखा पुस्तक के भाग IV और SID 0006 (revision note, Part H) से।

2.1 डिज़ाइन के चुनाव

नीचे के अधिकांश विचार शोध साहित्य या पहले की प्रणालियों से आते हैं, और पुस्तक का भाग II उनकी परंपरा का विवरण देता है। हमने उन्हें एक सीमित अनुरोध पाथ में जोड़ने, हर चरण को सस्ता रखने और परिणाम को ईमानदारी से मापने का प्रयास किया है। हम हर चुनाव को उसके कारण के साथ सूचीबद्ध करते हैं।

1. ऐसे चैलेंज जिन्हें जारी करने में सर्वर का कुछ खर्च नहीं होता। चैलेंज कुंजी-युक्त हैश से सील किया गया एक स्वयं-प्रमाणित रिकॉर्ड है, इसलिए उसे जारी करने में कोई अवस्था नहीं लिखी जाती; केवल चुकाया गया समाधान मेमोरी घेरता है, और चैलेंज की अवधि समाप्त होने पर वह भी हट जाता है (एल्गोरिदम और उनकी गारंटियाँ)।
2. प्रति नियम चुने गए कार्य के दो टियर। Hashcash के साथ Sibuna Cohen-Pietrzak का अनुक्रमिक कार्य का प्रमाण भी देता है, जिसकी सुदृढ़ता क्वांटम प्रूवर के विरुद्ध भी सिद्ध हो चुकी है, और जिसके प्रमाण का सत्यापन माइक्रोसेकंड में होता है। checkout रूट अनुक्रमिक कार्य के 20 वर्क बिट्स माँग सकता है, जबकि कोई ब्लॉग Hashcash के 16 बिट (कार्य के प्रमाण: समानांतर खोज और अनुक्रमिक कार्य)।
3. ऐसे सत्र जो चुकाया गया कार्य याद रखते हैं। सत्र टोकन उस तंत्र और उन वर्क बिट्स को दर्ज करता है जिन्हें उसके धारक ने हल किया। वह केवल उन्हीं रूट पर प्रवेश देता है जो उससे अधिक नहीं माँगते, इसलिए सस्ता सत्र कोई महँगा रूट नहीं खोल सकता।
4. सममित टोकन। सत्र कुंजी-युक्त BLAKE3 से सील किए जाते हैं, जिसका सत्यापन 155.8 ns में होता है, यानी Ed25519 हस्ताक्षर से लगभग 260 गुना कम कार्य, और जिसके विरुद्ध क्वांटम खोज केवल द्विघाती गति-वृद्धि देती है।
5. ऐसा निरीक्षण जिसे सत्र बायपास नहीं कर सकता। प्रवेश और निरीक्षण एक ही पास में चलते हैं, और वैध सत्र केवल चैलेंज को संतुष्ट करता है: सत्र और `q=' OR 1=1--` वाला अनुरोध फिर भी अस्वीकार होता है।
6. रैखिक समय वाला, संरचना को समझने वाला निरीक्षण। एक byte-class स्कैन, ऑटोमेटन का एक पास और छोटे संरचनात्मक टोकनाइज़र इनपुट में रैखिक समय में निर्णय लेते हैं, बैकट्रैक कर सकने वाले रेगुलर एक्सप्रेशन के बिना।
7. कुंजी-युक्त स्थान-निर्धारण के साथ सीमित अवस्था। हर टेबल की एक निश्चित क्षमता और भर जाने पर एक घोषित व्यवहार है, और कुंजियाँ कुंजी-युक्त हैश से रखी जाती हैं, इसलिए हमलावर किसी पीड़ित पर टकराव (collision) नहीं साध सकता (कुंजी-युक्त हैशिंग और सुरक्षा समीक्षा)।
8. प्रति अनुरोध कोई आवंटन नहीं, और बिना रुकावट नीति अपडेट। एक कनेक्शन हर निर्णय एक निश्चित बफ़र से सर्व करता है, और नया नीति इंजन किसी अनुरोध या writer को प्रतीक्षा कराए बिना प्रकाशित होता है (Mechanical Sympathy: शून्य आवंटन और सीमित अवस्था)।
9. लोड के साथ चलने वाली कठिनाई। प्रति सेकंड 50 सत्यापित समाधानों की आधार रेखा से ऊपर, दर के हर दोगुना होने पर एक वर्क बिट जुड़ता है, अधिकतम छह तक, इसलिए बाढ़ घातीय रूप से अधिक चुकाती है, जबकि शांत ट्रैफ़िक आधार लागत पर ही रहता है।
10. घटनाओं से कैंपेन। अस्वीकृत अनुरोध एक छोटी एम्बेडिंग के साथ दर्ज होते हैं, और मिलती-जुलती घटनाएँ कैंपेन में समूहित होती हैं, जिनकी जाँच ऑपरेटर कंसोल से कर सकता है।

3 Mechanical Sympathy: शून्य आवंटन और सीमित अवस्था

> **Knuth, यांत्रिक सटीकता पर: cache line और ठोस गणित**
>
> जो प्रोग्रामर किसी real-time प्रणाली के भीतरी लूप में general-purpose एलोकेटर को कॉल करता है, उसने डिज़ाइन का एक हिस्सा किसी अजनबी को सौंप दिया है। एलोकेटर औसत रूप से तेज़ हो सकता है, लेकिन उसकी लागत हीप के इतिहास पर निर्भर करती है, और वह इतिहास वही लिखता है जो अनुरोध भेजता है। कुछ भी optimise करने से पहले हमें गिनना चाहिए: हर चरण कितने बाइट छूता है, कितनी cache line, कितने आश्रित लोड। फिर हमें मापना चाहिए, क्योंकि गिनती भी गलत हो सकती है। कोई प्रोग्राम तब सुरुचिपूर्ण होता है जब पहला सॉकेट खुलने से पहले उसकी ज़रूरत के हर बाइट का एक ज्ञात ठिकाना हो, और जब हर लूप ठीक उतना ही कार्य करे जितना उसका गणित माँगता है।

3.1 शून्य-आवंटन का अपरिवर्तनीय नियम

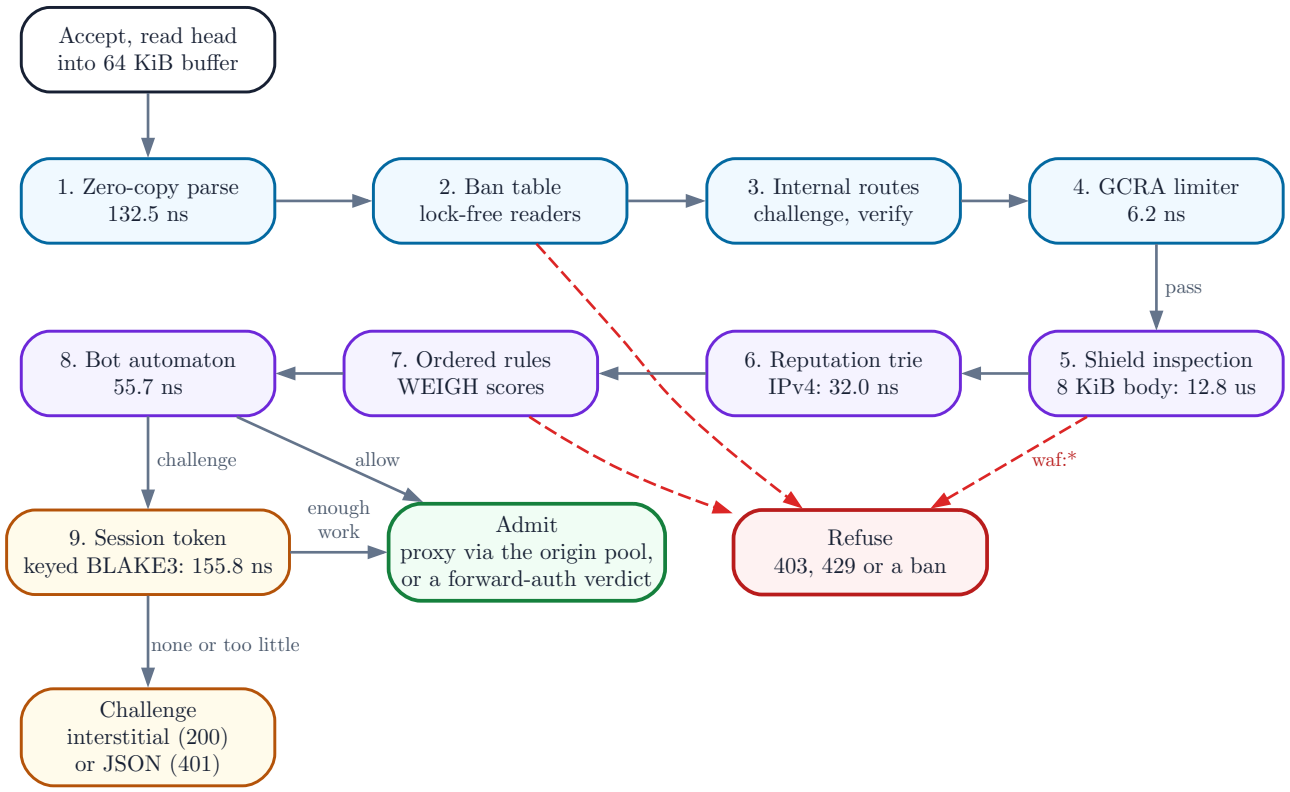
अनुरोध पाथ पर हर डायनामिक आवंटन denial of service का एक लीवर है: दिनों में बढ़ता विखंडन, एलोकेटर के लॉक पर होड़, और आवंटन कराने के लिए गढ़े गए अनुरोधों द्वारा प्रवर्धन। Sibuna का नियम यह है कि वर्गीकरण, सत्यापन और प्रॉक्सी को सौंपना हीप आवंटन के बिना चलते हैं। एक कनेक्शन अपने सॉकेट के पूरे जीवनकाल के लिए एक 64 KiB रीड बफ़र और एक 16 KiB राइट बफ़र का स्वामी है, और कनेक्शन शुरू होने पर एक बार आवंटित 40 KiB के प्रॉक्सी वर्कस्पेस का भी। बाकी सब उनमें स्लाइस है: पार्सर अनुरोध के मेथड, पाथ, क्वेरी, हेडर और कुकी को रीड बफ़र के भीतर ही खोजता है और उनमें से किसी को कॉपी नहीं करता[3]। इम्प्लीमेंटेशन का type system सुनिश्चित करता है कि पार्स किया गया अनुरोध उन बाइट से अधिक समय तक जीवित न रहे जिनका वह वर्णन करता है, इसलिए जब तक कोई अनुरोध बफ़र पढ़ रहा है, बफ़र दोबारा नहीं भरा जा सकता।

यह नियम अनुरोध पाथ का अनुबंध है, पूरी प्रोसेस के बारे में दावा नहीं। थ्रेड बनाना, स्टार्टअप और स्टोरेज आवंटन करते हैं। अनुबंध की जाँच आवंटन गिनने वाले परीक्षण करते हैं: डेमन का steady-state परीक्षण उसके आवंटन गिनता है, और प्रॉक्सी का परीक्षण 500 प्रॉक्सी किए गए keep-alive अनुरोधों में कोई आवंटन नहीं पाता[4]।

3.2 सीमित थ्रेड, दोबारा उपयोग

हर स्वीकृत कनेक्शन उसके अपने थ्रेड पर सर्व होता है, इसलिए धीमा क्लाइंट केवल स्वयं को धीमा करता है, और अनुरोध पाथ synchronous रहता है। हमने event loop पर विचार किया और थ्रेड मॉडल रखा: अनुरोध पाथ की गति zero-copy पार्सर, प्रति कनेक्शन निश्चित बफ़र और आवंटन-मुक्त निर्णयों से आती है, और event loop इनमें से किसी को बेहतर नहीं करता[5]। कनेक्शन थ्रेड की संख्या `--max-connections` (डिफ़ॉल्ट 1,024) से सीमित है; उसके आगे accept लूप कुछ भी शुरू किए बिना `503` उत्तर देता है। एक सॉकेट पीयर पता डिफ़ॉल्ट रूप से अधिकतम 64 कनेक्शन रख सकता है, और IPv6 पीयर अपने /64 नेटवर्क के साथ गिना जाता है।

[3] पुस्तक का भाग V, "प्रति कनेक्शन एक थ्रेड, सीमित" और "प्रति कनेक्शन एक बफ़र"।
[4] पुस्तक का भाग V, "कनेक्शन लूप"; `cmd/sibuna/tests/steady_state.rs` और `crates/net/tests/steady_state.rs`।
[5] SID 0010, "Concurrency model" और "Rationale and alternatives"; SID 0002, "Request contract" और उसके revision नोट (प्रति पता कनेक्शन, IPv6 प्रीफ़िक्स, striped ऑरिजिन पूल)।



चित्र 2 अनुरोध पाइपलाइन, जहाँ प्रिमिटिव सूट मापता है वहाँ एक कोर पर हर प्रिमिटिव के माध्यिका समय के साथ (`benchmarks/results/primitives-latest.md`)। चरण क्रमांकित क्रम में चलते हैं; डैश वाले तीर जल्दी अस्वीकृति दिखाते हैं (GCRA लिमिटर और प्रतिष्ठा trie भी इसी तरह अस्वीकार करते हैं)। ब्राउज़र अनुरोध का पूरा वर्गीकरण Gate प्रोफ़ाइल में 75.9 ns और Shield प्रोफ़ाइल में 790.8 ns लेता है।

थ्रेड join किए जा सकने वाले स्लॉट की एक निश्चित टेबल से आते हैं। समाप्त हुआ थ्रेड join होने तक अपना स्टैक रखता है, इसलिए नया task कर्सर आगे बढ़ाने के बजाय सबसे निचले समाप्त स्लॉट को join करके लेता है; तब resident मेमोरी सर्व किए गए नहीं, खुले कनेक्शनों के साथ चलती है। जो थ्रेड अपना कनेक्शन पूरा कर लेता है, वह रुककर अगला ले लेता है। अनुभाग मापन ने क्या बदला बताता है कि एक मापन ने हमें यह अंतिम नियम जोड़ना कैसे सिखाया।

3.3 अनुरोध पाइपलाइन

अनुरोध नीचे की जाँचों से एक निश्चित क्रम में गुज़रता है। सस्ती जाँचें पहले आती हैं, और अनुरोध को समाप्त करने वाला हर निर्णय किसी अधिक महँगे चरण के चलने से पहले ही ले लिया जाता है।

वैध सत्र चैलेंज को संतुष्ट करता है; वह कभी किसी अस्वीकृति या WAF निष्कर्ष को ओवरराइड नहीं करता। प्रॉक्सी, अपनी ओर से, ऑरिजिन का रिस्पॉन्स हेड पढ़ता है, उसकी फ़्रेमिंग वर्गीकृत करता है (कोई बॉडी नहीं, एक लंबाई, chunked coding, या बंद होने तक), बॉडी को ठीक-ठीक रिले करता है, और साफ़ सीमा पर छूटे ऑरिजिन सॉकेट को 256 सॉकेट के एक striped पूल में लौटा देता है, ताकि प्रॉक्सी किया गया क्लाइंट कनेक्शन खुला रहे और हर अनुरोध के लिए ऑरिजिन से दोबारा न जुड़ना पड़े।

3.4 अनुरोध रोके बिना नीति का प्रकाशन

नीतियाँ बदलने पर नीति इंजन अनुरोध पाथ से बाहर दोबारा बनाया जाता है, और अनुरोधों को कभी आधा बना इंजन नहीं दिखना चाहिए। स्टोरेज थ्रेड एक पूरा इंजन बनाता है और उसे reference count के साथ, generation काउंटर वाले एक स्लॉट में प्रकाशित करता है। हर कनेक्शन एक cache रखता है: वह एक acquire load से generation पढ़ता है, और केवल किसी प्रकाशन के बाद ही नया reference लेता है। पुराना इंजन तब मुक्त होता है जब उसे रखने वाला अंतिम cache उसे छोड़ देता है, और कोई writer कभी पाठकों के खाली होने की प्रतीक्षा नहीं करता। होड़ में cache किया गया रीड read-write लॉक लेने से 400 गुना से अधिक सस्ता है। इंजन स्वयं लगभग 200 KiB की सघन टेबल (209,052 बाइट) रखता है, जिसमें 128 नियम और 8,192 trie नोड की क्षमता है।

3.5 शार्ड में लॉक

कनेक्शन थ्रेड जिन टेबल को साझा करते हैं (spent set, लिमिटर, बैन टेबल), वे कुंजी के हैश से 16 शार्ड में बँटी हैं, और हर शार्ड अपनी अलग cache line पर है। उनके critical section दसियों निर्देश लंबे हैं, इसलिए हमने spin lock की तुलना adaptive mutex से मापी, जो थोड़ी देर spin करता

तालिका 2 Ryzen 9 5950X पर, लूप में पढ़ते 1 और 16 थ्रेड के साथ एक इंजन रीड की लागत (पुस्तक का भाग V, अक्टूबर 2026)

पाठक	1 थ्रेड	16 थ्रेड
read-write लॉक, केवल read guard	4.2 ns	752 ns
read-write लॉक और एक नया reference	7.6 ns	833 ns
cache किए गए reference के साथ generation जाँच	0.8 ns	1.8 ns

तालिका 3 Ryzen 9 5950X पर शार्ड लॉक का थ्रूपुट, प्रति सेकंड लाखों ऑपरेशन में (पुस्तक का भाग V; SID 0010)

वर्कलोड	adaptive mutex	Spin lock
spent set, 16 थ्रेड	10.8	8.6
spent set, 64 थ्रेड	11.2	6.9
छह व्यस्त पतों पर GCRA, 16 थ्रेड	44.9	19.3
छह व्यस्त पतों पर GCRA, 64 थ्रेड	37.4	6.5

है और फिर सो जाता है। जब कोई होड़ नहीं होती तो spin lock लगभग एक नैनोसेकंड बचाता है, लेकिन edge सर्वर CPU से अधिक थ्रेड चलाता है, और जिस spin lock का धारक preempt हो जाए वह हर प्रतीक्षक से उसका time slice जलवाता है। होड़ में mutex जीता, इसलिए हमने उसे रखा।

4 एल्गोरिदम और उनकी गारंटियाँ

अनुरोध पाथ के हर प्रिमिटिव के लिए यह कथन है कि उसकी लागत क्या है और वह क्या गारंटी देता है। उनमें से तीन को हम प्रमेय के रूप में, संक्षिप्त तर्क के साथ, देते हैं; SID 0006 और Sibuna पुस्तक के भाग III से VI पूरे प्रमाण, परिभाषाएँ और संदर्भ देते हैं।

प्रमेय 1 (byte-class Aho-Corasick ऑटोमेटन से रैखिक-समय निरीक्षण). मान लें $\mathcal{P}$ कुल लंबाई m वाले पैटर्न का एक समुच्चय है जिसके बाइट $c \le 256$ class में आते हैं, और किसी टेक्स्ट की लंबाई M है। Sibuna ऑटोमेटन को $O(mc)$ समय और स्थान में बनाता है और $O(M)$ समय में बताता है कि टेक्स्ट में कोई पैटर्न आता है या नहीं, प्रति इनपुट बाइट एक आश्रित टेबल लोड के साथ, पैटर्न की संख्या से स्वतंत्र। कोई भी इनपुट इसे बैकट्रैक नहीं करा सकता।

प्रमाण की रूपरेखा। Aho और Corasick का ऑटोमेटन failure link वाला trie बनाने के बाद टेक्स्ट को एक बार स्कैन करता है। Sibuna बिल्ड के समय failure link को ट्रांज़िशन टेबल में समाहित कर देता है, इसलिए हर इनपुट बाइट एक ट्रांज़िशन है और कभी कोई failure शृंखला नहीं चलनी पड़ती। byte class या तो किसी पैटर्न में आने वाला बाइट है, जिसमें किसी अक्षर के दोनों case एक class में होते हैं, या बाकी सभी बाइट की एकल class; इसलिए टेबल की एक पंक्ति में 256 के बजाय c प्रविष्टियाँ होती हैं। मैच अवस्थाएँ बाकी सभी अवस्थाओं के बाद क्रमांकित होती हैं, इसलिए मैच का पता लगाना एक तुलना है।

मापन। 40 बॉट सिग्नेचर पर ऑटोमेटन 55.7 ns लेता है, जबकि उन्हीं पैटर्न और any-match अर्थ के साथ अनुक्रमिक सबस्ट्रिंग खोज 763.0 ns लेती है, यानी 13.7 गुना तेज़। byte class के साथ पूरी बॉट सिग्नेचर टेबल 42,244 बाइट लेती है[6]।

□

प्रमेय 2 (GCRA अंतराल सीमा). प्रति विंडो W में N अनुरोधों की सीमा तय करें, उत्सर्जन अंतराल $T = \max(1, \lceil W/N \rceil)$ और बर्स्ट टॉलरेंस $\tau = (N-1)T$ के साथ। एक क्लाइंट एक सैद्धांतिक आगमन समय TAT रखता है। समय t पर आगमन अनुरूप है यदि और केवल यदि $\mathrm{TAT} \le t + \tau$, जिसके बाद $\mathrm{TAT} \leftarrow \max(\mathrm{TAT}, t) + T$। तब किसी अनुरूप आगमन से शुरू होने वाले L लंबाई के किसी भी अंतराल में अधिकतम $N + \lfloor L/T \rfloor$ अनुरूप आगमन होते हैं।

प्रमाण की रूपरेखा। अनुरूप आगमनों को $t_1 \le \ldots \le t_k$ समयों पर $1, \ldots, k$ क्रमांक दें। हर अनुरूप आगमन TAT को कम से कम T बढ़ाता है, इसलिए j-वें के बाद $\mathrm{TAT}_j \ge t_1 + jT$। आगमन k अनुरूप था, इसलिए $\mathrm{TAT}_{k-1} \le t_k + \tau \le t_1 + L + \tau$। दोनों मिलकर $(k-1)T \le L + (N-1)T$ देते हैं, इसलिए $k \le N + L/T$, और k एक पूर्ण संख्या है।

इम्प्लीमेंटेशन। किसी क्लाइंट की अवस्था एक 16-बाइट सेल है, बिना प्रति-अनुरोध इतिहास और बिना निश्चित विंडो की सीमा के प्रभाव के। 16 शार्ड `--rate-table-cells` सेल साझा करते हैं (डिफ़ॉल्ट 262,144, 4 MiB), जो कुंजी-युक्त हैश से रखी जाती हैं, और खाली हो चुकी सेल बिना sweeper के उसी स्थान पर वापस ली जाती है। जिस क्लाइंट का हैश डेमन पहले ही निकाल चुका है, उस पर एक जाँच 6.2 ns लेती है[7]।

□

प्रमेय 3 (अवस्था केवल चुकाए गए कार्य के साथ बढ़ती है). मान लें S उन चैलेंज टैग का समुच्चय है जिन्हें डेमन याद रखता है। कोई टैग S में तभी आता है जब उसके चैलेंज के लिए वैध प्रमाण स्वीकार हो चुका हो, और चैलेंज की अवधि समाप्त होने पर निकल जाता है। यदि r एक चैलेंज अवधि में स्वीकृत अलग-अलग समाधानों की दर को सीमित करता है, तो $|\ S\ | \le r \cdot \mathrm{TTL}$। जो चैलेंज लाया गया लेकिन हल नहीं हुआ, वह कोई मेमोरी नहीं घेरता।

प्रमाण की रूपरेखा। चैलेंज पहचानकर्ता एक स्वयं-प्रमाणित रिकॉर्ड है: संस्करण, एल्गोरिदम, कठिनाई, ओपनिंग की संख्या, जारी करने का समय, क्लाइंट का कुंजी-युक्त फ़िंगरप्रिंट, pseudorandom फ़ंक्शन से निकाला गया नॉनस, और उसे माँगने वाले नियम का हैश; 36 बाइट, जिनके बाद 16-बाइट का कुंजी-युक्त BLAKE3 टैग आता है, URL-safe base64 में 70 अक्षर। इसे जारी करने में कुछ नहीं लिखा जाता। सत्यापन पहले स्थिर समय में टैग जाँचता है, फिर आयु और क्लाइंट बाइंडिंग, फिर प्रमाण, और उसके बाद ही टैग को spent set में डालता है। keyed मोड में BLAKE3 को pseudorandom फ़ंक्शन के रूप में मॉडल करने पर, q सत्यापन क्वेरी करने वाला विरोधी अधिकतम $q \cdot 2^{-128}$ और PRF लाभ के योग जितनी प्रायिकता से टैग जाली बना पाता है।

[6] `benchmarks/results/primitives-latest.md` (55.7 और 763.0 ns); टेबल का आकार `primitives-latest.json` से, जैसा पुस्तक के भाग VIII, "संसाधन उपयोग (Footprint)" में दिया है; SID 0006, Part F, और byte class पर उसका revision note।

[7] SID 0006, Part D (Definition 22, Theorem 23); SID 0004, revision नोट (कुंजी-युक्त स्थान-निर्धारण, `--rate-table-cells`); `benchmarks/results/primitives-latest.md` (`gcra_check_hashed`)।

इम्प्लीमेंटेशन। spent set Robin Hood open addressing के 16 शार्ड है। समाप्त प्रविष्टि को उसी स्थान पर केवल ऐसी कुंजी अधिलेखित कर सकती है जो अपने home स्लॉट से कम से कम उतनी ही दूर हो, अन्यथा बाद का कोई लुकअप जल्दी रुककर किसी जीवित टैग को चूक सकता है; इस नियम के साथ किसी sweeper श्रेड की ज़रूरत नहीं, और अस्वीकृत प्रविष्टि कभी किसी जीवित टैग को नहीं हटाती। एक spend और एक लुकअप मिलकर 21.0 ns लेते हैं[8]।

□

4.1 सिमैंटिक फ़ायरवॉल

Shield पाथ, क्वेरी, User-Agent, गैर-संरचनात्मक हेडर और बॉडी के पहले 8 KiB का निरीक्षण करता है। उसके पहले संस्करण ने हमें एक सबक सिखाया। वह हर हेडर पर सबस्ट्रिंग की एक सपाट सूची का मिलान करता था, सूची में `/*` था, और हर ब्राउज़र `Accept: text/html,...,*/*;q=0.8` भेजता है। हर वास्तविक ब्राउज़र SQL injection के रूप में वर्गीकृत हुआ। तब से कोई डिटेक्टर केवल संरचना पर सक्रिय हो सकता है, और एक परीक्षण एक पूरा Chrome अनुरोध इंजन से गुज़ारता है और माँग करता है कि वह साफ़ निकले[9]।

निरीक्षण के चार चरण हैं, और हर चरण फ़ील्ड में रैखिक है।

1. *byte-class* प्री-स्कैन। 256 प्रविष्टियों वाली टेबल के साथ एक पास दर्ज करता है कि बाइट की कौन-सी class आती हैं: quote, `=`, `<`, shell separator, `%`, NUL और canonicalisation के ट्रिगर। बाद का हर डिटेक्टर दोबारा स्कैन करने के बजाय इन class से परामर्श करता है, इसलिए quote या बराबर के चिह्न के बिना कोई फ़ील्ड कभी SQL टोकनाइज़र में नहीं जाता।
2. टैग किया गया सिग्नेचर ऑटोमेटन। हर श्रेणी के सभी मज़बूत सिग्नेचर एक ऑटोमेटन में रहते हैं (प्रमेय 1)। हिट एक अंतिम अस्वीकृति है, जिसका नाम `waf:sqli`, `waf:xss`, `waf:path-traversal` या `waf:rce` होता है।
3. संरचनात्मक टोकनाइज़र। SQL टोकनाइज़र हर छोटे शब्द को कीवर्ड टेबल में खोजता है और `or` तथा `and` के बाद `literal = literal` tautology की जाँच करता है; अन्यथा quote अनिवार्य है और साक्ष्य को एक छोटे स्कोर तक पहुँचना होता है। वाक्यांश "it's a group order from the shop" का स्कोर 3 है और वह पास हो जाता है। Cross-site scripting के लिए स्क्रिप्ट रख सकने वाला टैग और एट्रिब्यूट की स्थिति में event-handler एट्रिब्यूट, दोनों चाहिए, और command injection के लिए ऐसा separator चाहिए जिसके बाद कोई कमांड नाम हो जो assignment न हो, इसलिए `a=1&id=5` एक पैरामीटर है, कमांड नहीं।
4. *Canonicalisation*। percent escape, कमेंट ओपनर, JSON या HTML escape, या समेटे जा सकने वाले white space वाला फ़ील्ड डिकोड करके दोबारा निरीक्षित किया जाता है, ताकि `1%27/**/UnIoN/**/SeLeCt` और `%252e%252e/` कच्चे सिग्नेचर पर सिमट जाएँ।

8 KiB बॉडी 12,784.3 ns में स्कैन होती है, लगभग 1.56 ns प्रति बाइट। ऑपरेटर किसी श्रेणी को enforce करने के बजाय ऑडिट मोड में रख सकता है, ताकि उसके निष्कर्ष लॉग हों लेकिन उन पर कार्रवाई न हो। तब फ़ील्ड का निर्णय प्रति श्रेणी करना होता है, ताकि कोई ऑडिट किया गया निष्कर्ष किसी enforce किए गए निष्कर्ष को छिपा न सके। फ़ील्ड को प्रति श्रेणी एक बार स्कैन करने के बजाय Sibuna उसे उन श्रेणियों के समुच्चय के लिए एक बार स्कैन करता है जिनके सिग्नेचर आते हैं, फिर उनमें से केवल वांछित श्रेणियाँ खोजता है; SQL injection ऑडिट मोड में होने पर वही 8 KiB बॉडी 15,485.4 ns लेती है[10]। Shield एक सीमित ह्यूरिस्टिक निरीक्षक बना रहता है, पूरा SQL या HTML पार्सर नहीं, और निरीक्षण की सीमा से आगे के बाइट का निरीक्षण नहीं होता।

4.2 इनग्रेस से TLS फ़िंगरप्रिंट

User-Agent एक ऐसी स्ट्रिंग है जिसे कोई भी क्लाइंट अपनी इच्छा से लिख सकता है। JA4 फ़िंगरप्रिंट को जाली बनाना कठिन है। वह संक्षेप में बताता है कि कोई क्लाइंट TLS कनेक्शन कैसे खोलता है: प्रोटोकॉल, संस्करण, cipher suites, extensions और पहला ALPN मान। क्लाइंट की TLS लाइब्रेरी इन्हें पहला HTTP बाइट भेजे जाने से पहले तय कर देती है। केवल वही घटक handshake देखता है जो TLS टर्मिनेट करता है, इसलिए केवल वही JA4 की गणना कर सकता है। Sibuna कोई आने वाला TLS टर्मिनेट नहीं करता, इसलिए वह JA4 की गणना नहीं करता। वह उस फ़िंगरप्रिंट को पढ़ता है जिसे किसी विश्वसनीय इनग्रेस ने गणना करके `--ja4-header` से नामित अनुरोध हेडर में फ़ॉरवर्ड किया[11]।

- भरोसा। हेडर केवल तभी पढ़ा जाता है जब फ़ॉरवर्ड किए गए हेडर पर भरोसा हो, उसी निर्णय से जो `X-Forwarded-For` को नियंत्रित करता है। अविश्वसनीय पीयर से आया हेडर अनदेखा किया जाता है, और रिवर्स-प्रॉक्सी मोड में अनुरोध के ऑरिजिन तक पहुँचने से पहले उसकी हर प्रति हटा दी जाती है।
- सत्यापन। मान केवल 36-बाइट JA4 रूप में स्वीकार होता है, और केवल तब जब उस नाम का ठीक एक फ़ील्ड मौजूद हो। विकृत या दोहराया गया मान अनुपस्थित माना जाता है और गिना जाता है।
- नीति। नियम पूरे फ़िंगरप्रिंट से, या ऐसे प्रीफ़िक्स से जिसके बाद `*` हो, मेल खाता है; फ़िंगरप्रिंट रहित अनुरोध कभी ऐसे नियम से मेल नहीं खाता। मिलान बाइट की तुलना है, और warm अनुरोध पाथ अब भी कोई हीप आवंटन नहीं करता।
- रिकॉर्ड। घटनाएँ अपने अनुरोध का फ़िंगरप्रिंट संग्रहीत करती हैं, और कंसोल उसे घटना के साक्ष्य में दिखाता है और उससे इवेंट फ़िल्टर करता है। प्रतिबंध और प्रतिष्ठा पते से कुंजीबद्ध रहते हैं, क्योंकि कई वैध क्लाइंट एक ही TLS लाइब्रेरी साझा करते हैं।

4.3 कुंजी-युक्त हैशिंग और सुरक्षा समीक्षा

अक्टूबर 2026 में अनुरोध पाथ की एक प्रतिकूल (adversarial) समीक्षा ने उन तंत्रों में कमज़ोरियाँ पाईं जिन्हें हम तय मान चुके थे। हमने हर एक को ठीक किया और सुधार को उस तंत्र के डिज़ाइन रिकॉर्ड में दर्ज किया[12]।

[8]पुस्तक का भाग III (लेम्मा 3 और लेम्मा 4, वायर प्रारूप); पुस्तक का भाग V, "Robin Hood spent set"; SID 0006, Part E (Lemma 28); `benchmarks/results/primitives-latest.md` (`robin_hood_spend_and_lookup`)।

[9]SID 0004, "Revision note (2026-09-07)" और "Verification"; पुस्तक का भाग VI, "False positive से एक सबक"।

[10]`benchmarks/results/primitives-latest.md` (`8kb_body_semantic_scan`, `8kb_body_sqli_audit_other_enforce`); SID 0004, revision note "inspection modes"; पुस्तक का भाग VIII, "WAF निरीक्षण"।

[11]SID 0014, "Specification" और "Security and operational considerations"; पुस्तक का भाग IX, "इनग्रेस से JA4 फ़िंगरप्रिंट"; `.github/release-notes.txt`।

[12]SID 0002, revision नोट (फ़ॉरवर्ड की गई पहचान; कुकी के मान); SID 0004, revision नोट (कुंजी-युक्त स्थान-निर्धारण, बैन टेबल, मापी गई लागतें, शून्य से भरी टेबल); SID 0006, revision नोट (spent set की क्षमता, अनुकूली नियंत्रक का इनपुट); SID 0010, review history (सुरक्षा समीक्षा, 2026-10-15); `benchmarks/results/primitives-latest.md` (`gcra_check` और `gcra_check_hashed`)।

- अनुमान योग्य टेबल स्थान-निर्धारण। लिमिटर और बैन टेबल कुंजियों को स्रोत कोड में लिखे seed के तहत एक हैश से रखते थे, इसलिए कोई भी ऐसे पते निकाल सकता था जो किसी पीड़ित की probe विंडो साझा करें। अब स्थान-निर्धारण ऑपरेटिंग सिस्टम की यादृच्छिकता से ली गई कुंजी के तहत SipHash-1-3 का उपयोग करता है, और हर टेबल अपनी कुंजी स्वयं निकालती है। डेमन प्रति अनुरोध क्लाइंट का एक बार हैश करता है (8.7 ns) और वह हैश हर टेबल को देता है; अनुरोध का स्टोर पाथ 43 से घटकर 26 ns रह गया।
- प्रतिबंध का निष्कासन। प्रभावी प्रतिबंध कभी हटाया नहीं जाता। जब किसी कुंजी के लिए संभव हर स्लॉट में एक जीवित प्रतिबंध हो, तो नया प्रतिबंध अस्वीकार करके गिना जाता है, इसलिए प्रतिबंधित क्लाइंट अपना ही प्रतिबंध नहीं हटवा सकता।
- फ़ॉरवर्ड की गई पहचान। सबसे बाईं `X-Forwarded-For` प्रविष्टि क्लाइंट लिख सकता है। Sibuna दाईं ओर से `--trusted-hops` स्थान वाली प्रविष्टि लेता है, जिसे निकटतम विश्वसनीय प्रॉक्सी ने जोड़ा था।
- *Cookie tossing*। किसी अनुरोध में सत्र कुकी के कई मान हो सकते हैं; डेमन अधिकतम आठ आज़माता है, इसलिए किसी sibling domain द्वारा लगाई गई कुकी आगंतुक को चैलेंज के चक्र में नहीं फँसा सकती।
- *spent set* की क्षमता। 65,536 टैग वाली टेबल पाँच मिनट की चैलेंज अवधि में प्रति सेकंड लगभग 220 सत्यापित समाधानों से भर जाती थी, जिसके बाद ईमानदार समाधान अस्वीकार होने लगते थे। अब क्षमता कॉन्फ़िगर की जा सकती है, डिफ़ॉल्ट 1,048,576 टैग (24 MiB)।
- अनुकूली कठिनाई। लोड में कार्य बढ़ाने वाला नियंत्रक अब जारी किए गए चैलेंज के बजाय सत्यापित समाधान गिनता है, क्योंकि चैलेंज जारी कराने में बाढ़ भेजने वाले का कुछ खर्च नहीं होता।

कुंजी-युक्त स्थान-निर्धारण की एक कीमत है, और हम उसे प्रति टेबल नहीं, प्रति अनुरोध एक बार चुकाते हैं। कच्चे क्लाइंट बाइट का हैश स्वयं करने वाली लिमिटर जाँच 12.8 ns लेती है; हैश किए गए क्लाइंट पर जाँच, जिस पाथ का डेमन उपयोग करता है, 6.2 ns लेती है। चूँकि लिमिटर और spent set शून्य से भरकर आवंटित होते हैं, निष्क्रिय टेबल resident नहीं होतीं, इसलिए उनकी कॉन्फ़िगर की गई क्षमता उपयोग होने तक कोई मेमोरी नहीं लेती।

5 कार्य के प्रमाण: समानांतर खोज और अनुक्रमिक कार्य

5.1 Hashcash टियर

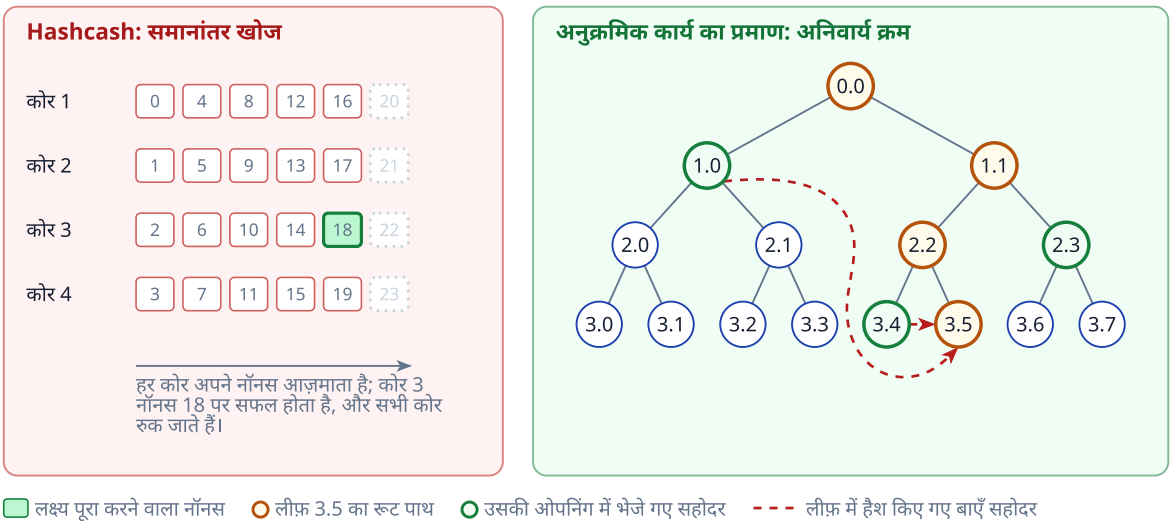
पहला टियर बिट-स्तर का Hashcash है। चैलेंज स्ट्रिंग C और कठिनाई b के लिए क्लाइंट ऐसा नॉनस N खोजता है कि SHA-256 ($C \,\|\, \texttt{":"} \,\|\, \mathrm{dec}\,(N)$) में b शुरुआती शून्य बिट हों। नॉनस का दशमलव रूप WebAssembly सॉल्वर, JavaScript फ़ॉलबैक और नेटिव सत्यापनकर्ता के बीच वायर प्रारूप को एक जैसा रखता है। आदर्श डाइजेस्ट के लिए प्रयासों की संख्या K, $p = 2^{-b}$ के साथ ज्यामितीय है:

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

खोज की दो सीमाएँ हैं। उसका प्रसरण बड़ा है, इसलिए एक धीमा हल खराब वर्कर का संकेत नहीं। और स्वतंत्र प्रयास समानांतर चलते हैं, इसलिए अधिक कोर वाला प्रेषक जल्दी पूरा करता है, भले ही अपेक्षित कुल कार्य वही हो[13]।

5.2 अनुक्रमिक कार्य का टियर

दूसरा टियर Cohen और Pietrzak (EUROCRYPT 2018) के अनुक्रमिक कार्य के प्रमाण का अनुसरण करता है। इसके लेबल गहराई n का एक पूर्ण बाइनरी ट्री बनाते हैं। आंतरिक नोड अपने दो चाइल्ड का हैश करता है। लीफ़ उन सभी स्थानों पर अपने पूर्वजों के बाएँ सहोदरों का हैश करती है जहाँ उसका रूट पाथ दाईं ओर मुड़ा। ये बाएँ-सहोदर किनारे गणना को अनुक्रमिक बनाते हैं: किसी लीफ़ का लेबल तब तक नहीं बन सकता जब तक उसके बाईं ओर का हर सबट्री पूरा न हो जाए, इसलिए रूट लेबल क्रम से सभी $2^{n+1} - 1$ लेबल पर निर्भर करता है। रूट φ की गणना के बाद प्रूवर Fiat-Shamir से t लीफ़ निकालता है और हर एक के लिए लीफ़ लेबल तथा उसके पाथ पर पड़ने वाले n सहोदर लेबल भेजता है। सत्यापनकर्ता हर खोली गई लीफ़ की दोबारा गणना करता है और φ तक हैश करता है, एक निश्चित वर्कस्पेस में $O(tn)$ लेबल गणनाओं की लागत पर।



चित्र 3 समानांतर खोज बनाम अनुक्रमिक कार्य। बाईं ओर, Hashcash: स्वतंत्र नॉनस प्रयास, इसलिए M कोर लगभग M गुना जल्दी पूरा करते हैं। दाईं ओर, गहराई 3 पर Cohen-Pietrzak ट्री: डैश वाले बाएँ-सहोदर किनारे लीफ़ 3.5 को उसके बाईं ओर के सबट्री की प्रतीक्षा कराते हैं, और लीफ़ 3.5 की ओपनिंग उसका लेबल और तीन हरे सहोदर भेजती है (पुस्तक के भाग III के आधार पर)।

[13] पुस्तक का भाग III, "Hashcash: टियर एक का गणित"; पुस्तक का भाग I, "यादृच्छिक चर के रूप में हैश खोज"।

पैरामीटर इस तरह चुने गए हैं कि संख्याएँ छोटी रहें। वर्क बिट्स $n = b - 3$ से गहराई में बदलते हैं, $[4, 24]$ में सीमित। एक प्रमाण में $32(1 + t(n + 1))$ बाइट होते हैं, जो 16 ओपनिंग के साथ गहराई 13 पर $32(1 + 16 \times 14) = 7200$ बाइट है। प्रूवर ट्री के केवल ऊपरी स्तर रखता है और जिस सबट्री को खोलना है उसकी दोबारा गणना करता है, इसलिए प्रमाण सहित उसका पूरा वर्कस्पेस 100 KB से कम रहता है; ब्राउज़र टैब और नेटिव परीक्षण एक ही प्रूवर का उपयोग करते हैं[14]। Blocki, Lee और Zhou (ITC 2021) ने quantum random-oracle model में इस निर्माण को क्वांटम प्रूवर के विरुद्ध सुदृढ़ सिद्ध किया।

हमें सावधान रहना चाहिए कि इससे क्या स्थापित होता है। सुरक्षा परिणाम शोध-पत्रों की random-oracle धारणाओं के तहत, उनमें बताए गए निर्माणों और सीमाओं के लिए सही हैं। वे हमारे ठोस SHA-256 प्रोटोकॉल, उसकी पैरामीटर मैपिंग या उसके ब्राउज़र समय का स्वतंत्र ऑडिट नहीं हैं। प्रारंभिक सैंपलिंग सीमा $(1 - \alpha)^t$, यानी यह संभावना कि t स्वतंत्र ओपनिंग किसी निश्चित खराब अंश α को पूरी तरह चूक जाएँ, समझाती है कि अधिक ओपनिंग क्यों मदद करती हैं; यह अपने आप में कई रूट आज़माने (grind करने) वाले प्रूवर के विरुद्ध प्रमाण नहीं है। हम कोई बिना शर्त दावा नहीं करते कि धोखा देना कभी सस्ता नहीं होता।

5.3 राउंड ट्रिप और सत्र

वैध सत्र के बिना कोई नेविगेशन इंटरस्टिशियल पेज पाता है, जो एक आवश्यकता टिकट ले जाता है: इस निर्णय द्वारा माँगा गया चैलेंज (एल्गोरिदम, वर्क बिट्स, ओपनिंग) और नियम का हैश, इस क्लाइंट के लिए सील किया गया और एक चैलेंज अवधि तक वैध। पेज टिकट के साथ अपना चैलेंज लाता है, इसलिए कार्य उसी निर्णय से तय होता है जिसने उसे माँगा। एक Web Worker चैलेंज को WebAssembly प्रूवर से हल करता है, या जहाँ WebAssembly उपलब्ध न हो वहाँ JavaScript फ़ॉलबैक से, और उत्तर पोस्ट करता है। सफलता पर सर्वर 75 अक्षरों की एक सत्र कुकी सेट करता है, जो कुंजी-युक्त फ़िंगरप्रिंट से क्लाइंट के पते और User-Agent से बँधी होती है, और पेज दोबारा लोड होता है। ब्राउज़र का प्रूवर उसी स्रोत से कंपाइल होता है जिससे सर्वर का सत्यापनकर्ता, इसलिए दोनों अलग-अलग राह नहीं पकड़ सकते[15]।

5.4 सत्यापन की लागत

दोनों टियर SHA-256 से बँधे हैं, और Sibuna जहाँ उपलब्ध हों वहाँ प्रोसेसर के SHA निर्देश रन टाइम पर चुनता है, इसलिए एक ही बाइनरी हर उस मशीन पर तेज़ है जिसमें वे निर्देश हैं। 16 बिट पर Hashcash सत्यापन 70.2 ns लेता है, और 16 ओपनिंग वाले गहराई-13 प्रमाण का सत्यापन 22,272.4 ns। ब्राउज़र प्रूवर 14,744 बाइट का एक WebAssembly मॉड्यूल है। end-to-end ब्राउज़र जाँच में headless Chromium ने इंटरस्टिशियल का चैलेंज WebAssembly के साथ 29 से 34 ms में और JavaScript फ़ॉलबैक के साथ 84 से 91 ms में हल किया[16]। प्रिमिटिव सूट केवल सत्यापन मापता है; नेटिव समय-माप ब्राउज़र के हल-समय का प्रतिनिधि नहीं है।

सत्र एक सममित टोकन का उपयोग करते हैं। सत्यापनकर्ता ही जारीकर्ता है, या एक सीड साझा करने वाला क्लस्टर, इसलिए कुंजी-युक्त pseudorandom फ़ंक्शन ठीक वही प्रमाणीकरण देता है जिसकी ज़रूरत है। कुंजी-युक्त BLAKE3 टोकन का सत्यापन 155.8 ns में और Ed25519 टोकन का 40,587.5 ns में होता है, इसलिए एक हस्ताक्षर-सत्यापन की लागत लगभग 260 MAC सत्यापनों के बराबर है। इस चुनाव का एक दूसरा कारण भी है: Shor का एल्गोरिदम Ed25519 के नीचे के discrete logarithm को तोड़ देता है, जबकि कुंजी-युक्त हैश के विरुद्ध क्वांटम खोज अधिकतम द्विघाती गति-वृद्धि देती है। जिन डिप्लॉयमेंट को टोकन न बना सकने वाले सत्यापनकर्ता चाहिए, उनके लिए Ed25519 एक विकल्प बना रहता है।

6 अपने किसी consensus प्रोटोकॉल के बिना साझा अवस्था

> **Lamport, वितरित अपरिवर्तनीय नियमों पर: Safety, Liveness, और लॉग किसके ज़िम्मे**
>
> 1987 में Leslie Lamport ने वितरित प्रणाली का वर्णन ऐसी प्रणाली के रूप में किया था जिसमें किसी ऐसे कंप्यूटर की विफलता, जिसके अस्तित्व का आपको पता भी नहीं था, आपके अपने कंप्यूटर को अनुपयोगी बना सकती है। सलाह के रूप में पढ़ें, तो यह हमसे कहता है कि हम उन कंप्यूटरों को गिनें जिन पर हम निर्भर हैं, और बताएँ कि उनमें से कोई भी विफल होने पर क्या सही रहना चाहिए। दो तरह के कथन पर्याप्त हैं। *Safety*: कभी कुछ बुरा नहीं होता, उदाहरण के लिए कोई नोड कभी ऐसी नीति पर कार्य नहीं करता जिसे किसी कमिट हुए ट्रांज़ैक्शन ने नहीं लिखा। *Liveness*: अंततः कुछ अच्छा होता है, उदाहरण के लिए डेटाबेस तक पहुँच सकने वाला नोड अंततः नवीनतम नीति सर्व करता है। consensus प्रोटोकॉल दोनों को पाने का एक सुंदर तरीका है। उसे अपने ज़िम्मे रखना एक बड़ा काम भी है, और फ़ायरवॉल के लिए प्रश्न यह नहीं कि consensus अच्छा है या नहीं, बल्कि यह कि लॉग का रखरखाव किसे करना चाहिए।

6.1 SQL क्यों, और अपना consensus प्रोटोकॉल क्यों नहीं

Sibuna कम ही संग्रहीत करता है: कुछ सौ नीति पंक्तियाँ, एक प्रतिष्ठा टेबल, और फ़ॉरेंसिक घटनाएँ, जिन्हें ऑपरेटर दूसरी प्रणालियों में भी export करते हैं। इस अवस्था को किसी विशेष रूप से बने replicated इंजन की ज़रूरत नहीं। उसे एक भरोसेमंद SQL स्टोर चाहिए, एक नोड के लिए स्थानीय और कई के लिए साझा। इसलिए एक अकेला नोड अपनी अवस्था अपनी डेटा डायरेक्टरी में WAL मोड वाली एम्बेडेड SQLite फ़ाइल में रखता है, और क्लस्टर हर नोड को एक PostgreSQL डेटाबेस की ओर इंगित करता है[17]।

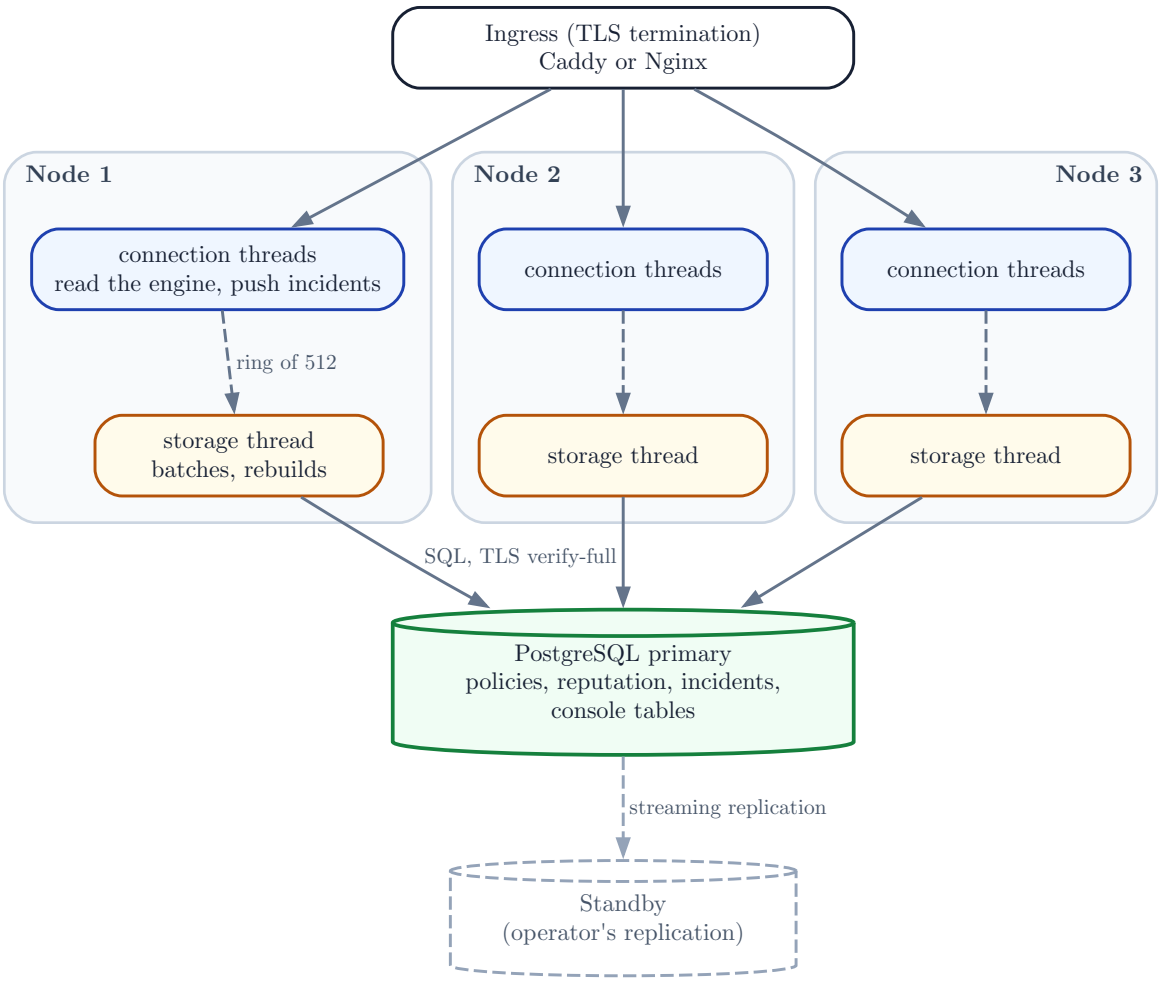
हमने फ़ायरवॉल के भीतर consensus प्रोटोकॉल न चलाना चुना। consensus इंजन फ़ायरवॉल के साथ ज़िम्मे रखने के लिए एक बड़ी प्रणाली है, अपने प्रारूपों और अपनी विफलताओं के ढंग के साथ, और वह शटडाउन तथा हर अपग्रेड में कार्य जोड़ता है। इसके बजाय क्लस्टर की अवस्था की उपलब्धता और रेप्लिकेशन PostgreSQL का विषय है, जिसे उसकी streaming replication और failover उपकरण या कोई managed सेवा संभालते हैं। हमें लगता है कि यह अधिक विनम्र चुनाव है: PostgreSQL के रेप्लिकेशन को हमारे संभावित डिप्लॉयमेंट से कहीं अधिक डिप्लॉयमेंट परख चुके हैं, और Sibuna की राइट दर (बैच में घटनाएँ और नीति के दुर्लभ बदलाव) इतनी कम है कि एक primary बाधा नहीं बनता।

[14] पुस्तक का भाग III, "अनुक्रमिक कार्य का प्रमाण: टियर दो", "प्रूवर की मेमोरी" और "पैरामीटर समय-माप नहीं हैं"; SID 0006, Part B और 2026-09-07 की उसकी implementation review।

[15] पुस्तक का भाग IV, "चैलेंज का राउंड ट्रिप" और "सत्र कुकी"; SID 0010, "The browser solver"।

[16] `benchmarks/results/primitives-latest.md`; WebAssembly का आकार `primitives-latest.json` से, जैसा पुस्तक के भाग VIII, "संसाधन उपयोग (Footprint)" में दिया है; ब्राउज़र के हल-समय SID 0010, criterion 3 (Chromium 155, `cargo xtask browser`) से; SHA निर्देशों का रन-टाइम पर पता लगाना SID 0006, revision note (Parts A to C) से।

[17] SID 0011, "Abstract", "Motivation and scope" और "Why PostgreSQL for clusters"; SID 0005 पहले के replicated डिज़ाइन का वर्णन करता है।



चित्र 4 एक Sibuna क्लस्टर। हर नोड अपना अनुरोध पाथ मेमोरी में रखता है और सारा SQL एक स्टोरेज थ्रेड को देता है; नोड एक PostgreSQL डेटाबेस साझा करते हैं, जिसका रेप्लिकेशन और failover PostgreSQL के ज़िम्मे है (SID 0011)। अकेला नोड इसके बजाय एम्बेडेड SQLite फ़ाइल का उपयोग करता है।

स्टोरेज वैकल्पिक भी रहता है। `--data-dir` या `--database-url` के बिना डेमन मेमोरी में चलने वाला फ़ायरवॉल है, जिसकी नीतियाँ एक फ़ाइल से आती हैं।

6.2 एक स्टोरेज थ्रेड

एक स्टोरेज थ्रेड डेटाबेस कनेक्शनों का स्वामी है; कोई और SQL को नहीं छूता। अनुरोध थ्रेड कभी उसकी प्रतीक्षा नहीं करते। वे घटना रिकॉर्ड 512 स्लॉट की एक सीमित रिंग में push करते हैं, जो भर जाने पर रिकॉर्ड छोड़कर गिन लेती है, और वे प्रकाशित इंजन पढ़ते हैं। डिफ़ॉल्ट रूप से हर 500 ms पर स्टोरेज थ्रेड एक tick चलाता है: वह कतार की अधिकतम 32 घटनाएँ एक ट्रांज़ैक्शन में लिखता है, नीति revision पढ़ता है और, बदलने पर, इंजन दोबारा बनाकर प्रकाशित करता है। दोबारा बनाना लोड करने से पहले और बाद में revision पढ़ता है; यदि बीच में किसी writer ने कमिट किया, तो उम्मीदवार छोड़ दिया जाता है और पुराना इंजन चलता रहता है। ऑपरेटर नीति टेबल में सीधे लिख सकते हैं: trigger writer के ट्रांज़ैक्शन में revision आगे बढ़ाते हैं, और PostgreSQL के अंतर्गत `NOTIFY` नोड को तुरंत tick करने देता है, जबकि polling गारंटी बनी रहती है[18]।

क्लस्टर के हर नोड का 1 से 8,388,607 के बीच एक अलग `--cluster-node` पहचानकर्ता होता है। चैलेंज कुंजियाँ जारी करने वाले नोड से बँधी हैं, इसलिए समाधान को उसी नोड पर लौटना होगा जिसने उसका चैलेंज जारी किया, जबकि सत्र टोकन और आवश्यकता टिकट सीड साझा करने वाले हर नोड पर वैध हैं। घटना पहचानकर्ता अपने ऊपरी बिट में नोड रखते हैं, और हर नोड एक स्थायी रसीद रखता है जो उसके बैच के ट्रांज़ैक्शन में ही आगे बढ़ती है, इसलिए अस्पष्ट विफलता के बाद दोबारा चलाया गया बैच कुछ भी दो बार नहीं लिखता।

स्टोरेज परत के अपरिवर्तनीय नियम (SID 0011). **S1 (एक स्वामी)।** सारा SQL स्टोरेज थ्रेड पर चलता है; कोई अनुरोध थ्रेड कभी डेटाबेस की प्रतीक्षा नहीं करता। **S2 (पूरे इंजन)।** हर इंजन जिसे कोई अनुरोध पढ़ता है, प्रकाशित होने से पहले नीति टेबल के एक सुसंगत revision से पूरी तरह बना था। **L1 (अंततः प्रकाशन)।** नीति टेबल में कमिट हुआ बदलाव डेटाबेस तक पहुँच सकने वाले हर नोड तक उसके अगले tick तक पहुँचता है। **F1 (सधी हुई हानि)।** डेटाबेस बंद होने पर अंतिम प्रकाशित इंजन सेवा देता रहता है, घटनाएँ सीमित रिंग में कतार लगाती हैं और भर जाने पर एक काउंटर के साथ छोड़ दी जाती हैं, और स्टोरेज थ्रेड backoff के साथ रीट्राई करता है।

6.3 प्रकारबद्ध (typed) स्कीमा

स्कीमा SQL में वही बताता है जो प्रोग्राम के प्रकार बताते हैं, इसलिए डेटाबेस ऐसी पंक्ति अस्वीकार करता है जिसे स्टोरेज परत व्यक्त नहीं कर सकती। SQLite टेबल `STRICT` हैं; गणनीय कॉलम पर ऐसा `CHECK` है जो ठीक उन्हीं नामों की सूची देता है जिन्हें प्रोग्राम लिखता है; सीमित पूर्णांक पर श्रेणी की

[18] SID 0011, "The storage thread", "Policy revision" और "Incidents and campaigns"।

जाँच है; और कॉलम ठीक तभी `NOT NULL` है जब मान कभी अनुपस्थित नहीं होता। हर स्टेटमेंट अपने मान पैरामीटर के रूप में bind करता है, इसलिए कोई SQL quoting से नहीं बनता। कंसोल का स्टोरेज भी इन्हीं नियमों का पालन करता है: 33 typed टेबल का एक baseline migration और कोई trigger नहीं। कंसोल का हर अपरिवर्तनीय नियम, जैसे "टोकन में हर बदलाव एक ऑडिट पंक्ति लिखता है", ठीक एक फ़ंक्शन द्वारा लागू होता है जो अपने स्टेटमेंट एक ट्रांज़ैक्शन में चलाता है, SQLite और PostgreSQL दोनों पर[19]।

जब डेमन `postgres-tls` feature के साथ बिल्ड होता है, तब PostgreSQL से कनेक्शन URL के अनुसार, libpq के `sslmode` और `sslrootcert` से, सुरक्षित होते हैं। इस feature के बिना बना बिल्ड स्टार्टअप पर `require`, `verify-ca` और `verify-full` को अस्वीकार करता है; वह कभी plaintext पर नहीं लौटता। प्रोडक्शन क्लस्टर `sslmode=verify-full` का उपयोग करते हैं।

मिलती-जुलती घटनाएँ कैंपेन में समूहित होती हैं। हर घटना 256 बाइट के रूप में संग्रहीत 64-आयामी एम्बेडिंग रखती है। स्टोरेज थ्रेड सभी नोड की सबसे हाल की 16,384 एम्बेडिंग मेमोरी में (4 MiB) रखता है और कोसाइन दूरी अधिकतम 0.35 होने पर घटना को उसके निकटतम पड़ोसी के कैंपेन में डालता है। पुरानी घटनाओं के विरुद्ध क्लस्टरिंग जानबूझकर छोड़ी गई है: कैंपेन स्वभाव से हाल के होते हैं, और यह सीमा मेमोरी और कार्य को निश्चित रखती है।

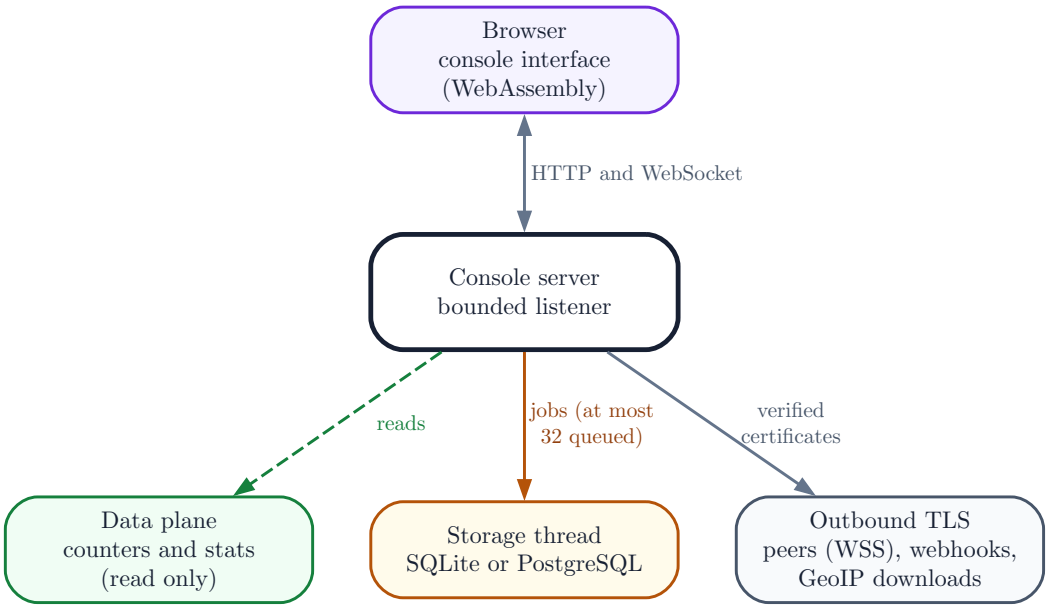
6.4 यह डिज़ाइन क्या नहीं करता

हमें इस चुनाव की सीमाओं के बारे में साफ़ होना चाहिए। Sibuna में अपना कोई consensus प्रोटोकॉल नहीं है, और क्लस्टर की सुसंगतता PostgreSQL के ज़िम्मे है: डेटाबेस खो देने वाला नोड दोबारा जुड़ने तक अपनी अंतिम नीति रखता है। एक नोड पर जारी सत्र हर नोड स्वीकार करता है, और एक नोड पर खर्च हुआ समाधान दूसरे नोड पर दोबारा भेजे जाने पर अस्वीकार होता है, क्योंकि नोड एक सीड साझा करते हैं और चैलेंज कुंजियाँ जारी करने वाले नोड से बँधी हैं; किसी भी गुण को डेटाबेस की ज़रूरत नहीं। SID 0011 committed है: एक PostgreSQL डेटाबेस पर तीन नोड वाला कार्यात्मक परीक्षण दिखाता है कि एक नोड पर डाली गई नीति एक tick के भीतर तीनों पर लागू होती है, एक नोड पर हनीपॉट प्रतिबंध दूसरों तक पहुँचता है, और घटना पहचानकर्ता कभी नहीं टकराते। अभी कोई हार्नेस लोड में एक PostgreSQL डेटाबेस पर कई नोड नहीं चलाता, इसलिए यह पत्र क्लस्टर का कोई मापन रिपोर्ट नहीं करता। रिलीज़ की बाइनरी PostgreSQL समर्थन के बिना बनती हैं; क्लस्टर Sibuna को स्रोत से `postgres-tls` feature के साथ बिल्ड करता है[20]।

7 कंसोल

Sibuna में एक वैकल्पिक प्रबंधन कंसोल है: लाइव ट्रैफ़िक वाले प्रमाणित डैशबोर्ड, किसी अस्वीकृति से उसके दर्ज साक्ष्य तक जाँच का रास्ता, नीति संपादन, पता समूह, खुले GeoIP डेटा से देश की जानकारी, Argon2id पासवर्ड और TOTP second factor वाले उपयोगकर्ता, ऑडिट ट्रेल, webhook, और क्लस्टर का प्रबंधन। यह उसी बाइनरी में कंपाइल होता है, और इसका इंटरफ़ेस ब्राउज़र में एक WebAssembly मॉड्यूल के रूप में चलता है[21]।

कंसोल की निर्णायक शर्त अनुरोध पाथ से पृथक्करण है। वह data plane के काउंटर और उसका डेटाबेस पढ़ सकता है और डेटाबेस में लिख सकता है, लेकिन वह कभी किसी अनुरोध थ्रेड में प्रवेश नहीं करता, कभी उस पर आवंटन नहीं करता, और कभी ऐसा लॉक नहीं पकड़ता जिसकी किसी कनेक्शन थ्रेड को ज़रूरत हो। इसका listener अधिकतम 96 कनेक्शन स्वीकार करता है, जिनमें से 16 HTTP के लिए आरक्षित हैं, 64 ब्राउज़र स्ट्रीम और 16 पीयर स्ट्रीम हो सकते हैं। 1.0.0-beta.3 में पूरा इंटरफ़ेस मॉड्यूल 724,330 बाइट का है (gzip से संपीड़ित 272,568 बाइट)। यह 768 KiB की ऊपरी सीमा



चित्र 5 कंसोल। ब्राउज़र इंटरफ़ेस को एक WebAssembly मॉड्यूल के रूप में चलाता है; कंसोल सर्वर data plane के काउंटर पढ़ता है और अपना SQL स्टोरेज थ्रेड को देता है; पीयर, webhook और GeoIP प्रकाशकों तक बाहर जाने वाले कनेक्शन सत्यापित प्रमाणपत्रों वाले TLS का उपयोग करते हैं (SID 0007, 0012, 0013)।

[19]SID 0011, "Schema and migrations" और उसकी review history (transport security); SID 0013, "Abstract"।

[20]SID 0011, "Security and operational considerations" और "Acceptance criteria and implementation status" (criterion 2, `crates/storage/tests/multi_node.rs`); `.github/release-notes.txt`, "Known limitations"; SID 0002, "Authentication and work" (नोड से बँधी चैलेंज कुंजियाँ); पुस्तक का भाग VIII, "जो मापा नहीं गया"।

[21]SID 0007, "Abstract" और उसके revision नोट (listener admission, interface module, GeoIP और पासवर्ड); SID 0010, review history (console port, TOTP); SID 0012; SID 0013। पृथक्करण का मापन SID 0010 की review history (console port) में है; पुस्तक का भाग VIII, "कंसोल का अतिरिक्त भार", बताता है कि रिकॉर्ड इसके लिए कोई परिणाम फ़ाइल नहीं बताता।

तालिका 4 headless Chromium 155 में, एक SQLite नोड पर उसके कंसोल के साथ आठ Traffic डैशबोर्ड (`benchmarks/results/console-8-dashboards-2026-10-15.md`, 2026-10-15 को दर्ज)। परिणाम: PASS

माप	मान
Data-plane अनुरोध	30.0 s में 6,000 (प्रति सेकंड 200, प्रति अनुरोध एक कनेक्शन)
उत्तर दिए गए	6,000: 200 x 1,200, 401 x 3,000, 403 x 600, 404 x 600, 500 x 600
त्रुटियाँ (कोई उत्तर नहीं)	0
लेटेंसी p50 / p90 / p99 / अधिकतम	246 / 311 / 373 / 552 us (connect सहित)
डेमन की resident मेमोरी: शुरुआत में, 8 डैशबोर्ड के साथ, ट्रैफ़िक के बाद	36,152 KiB; 38,028 KiB; 41,496 KiB
डैशबोर्ड	3 एडमिनिस्ट्रेटर सत्र और 5 wall display; हर एक को 134 से 139 `stats` delta मिले और सब एक जैसे योग पर समाप्त हुए, बिना किसी exception, कंसोल त्रुटि या Content-Security-Policy उल्लंघन के

से नीचे है, जिसे बिल्ड लागू करता है, और 640 KiB के समीक्षा थ्रेशोल्ड से ऊपर, जिसे SID 0007 समीक्षा के लिए दर्ज करता है[22]।

कंसोल लंबे समय तक चलने वाली पीयर स्ट्रीम, webhook और GeoIP डाउनलोड के लिए बाहर जाने वाले TLS कनेक्शन बनाता है। प्रमाणपत्र सिस्टम roots या ऑपरेटर की CA फ़ाइल के विरुद्ध सत्यापित होते हैं, होस्ट-नाम सत्यापन बंद नहीं किया जा सकता, सादा HTTP केवल लिटरल loopback पतों पर अनुमत है, और प्रमाणपत्र की त्रुटि कभी plaintext पर नहीं लौटती। Sibuna अब भी कोई आने वाला TLS टर्मिनेट नहीं करता; वह इनग्रेस का काम बना रहता है।

SID 0007 कंसोल का प्रदर्शन लक्ष्य तय करता है: उसे सक्षम करने की लागत data-plane थ्रूपुट का अधिकतम 1% और p99 लेटेंसी का अधिकतम 10% हो सकती है। एक मापन मौजूद है: कंसोल कंपाइल होने पर लेकिन अक्षम, केवल स्टोरेज के साथ, या सक्षम और निष्क्रिय होने पर, 16 पूरे-उत्पाद वर्कलोड उसके बिना बने डेमन के लगभग 1% के भीतर रहे, जो noise के भीतर है। लोड में आठ सक्रिय डैशबोर्ड वाला औपचारिक गेट नहीं चलाया गया है, इसलिए हम लक्ष्य पूरा होने का दावा नहीं करते। आठ लाइव डैशबोर्ड वाला एक कार्यात्मक रन पास हुआ।

8 प्रायोगिक मूल्यांकन

8.1 मेथड

`cargo xtask bench` के ज़रिए चलने वाले तीन हार्नेस Sibuna को मापते हैं। प्रिमिटिव सूट अनुरोध पाथ के ऑपरेशनों का समय बिना समय-माप वाले warm-up के बाद सात समय-मापित बैच में मापता है, परिवर्तनशील अवस्था को टाइमर के बाहर दोबारा बनाता है, और प्रति ऑपरेशन माध्यिका बैच को उसके साथ सबसे तेज़ और सबसे धीमे बैच के साथ रिपोर्ट करता है। पूरे-उत्पाद वाला हार्नेस डेमन को `wrk` (2 थ्रेड, 64 कनेक्शन, 5 सेकंड, 3 दोहराव) से forward-auth और रिवर्स-प्रॉक्सी मोड में, Gate और Shield सतहों पर, चार वर्कलोड पर चलाता है: `admitted` (वैध सत्र), `challenged` (कोई सत्र नहीं), `allowed_static` (`/robots.txt`) और `attack` (सत्र और साथ में `q=' OR 1=1--`, जिसे केवल Shield अस्वीकार करता है)। लेटेंसी हार्नेस एक open-loop जनरेटर का उपयोग करता है जो डेमन चाहे जो करे, एक निश्चित दर से भेजता है, और हर अनुरोध का समय उसके देय क्षण से मापता है, ताकि डेमन में लगी कतार डेमन के खाते में जाए[23]।

रन 15 अक्टूबर 2026 को एक होस्ट पर दर्ज किए गए: Linux 7.0.0-28 चलाता एक AMD Ryzen 9 5950X (16 कोर, 32 थ्रेड)। डेमन चार वर्कर के साथ CPU 0 से 3 पर चला, लोड जनरेटर और एक स्टब ऑरिजिन CPU 8 से 15 पर (दूसरा CCD), और प्रिमिटिव सूट CPU 4 पर। डेमन डिफ़ॉल्ट रिलीज़ बिल्ड था, SQLite और कंसोल के साथ कंपाइल हुआ, और स्टोरेज के बिना मेमोरी में चला।

> **मापन का दायरा.** हर रन loopback पर एक होस्ट का उपयोग करता है: कोई TLS नहीं, कोई wide-area नेटवर्क नहीं, कोई वास्तविक ब्राउज़र नहीं। हार्नेस आवंटन नहीं गिनते; आवंटन-मुक्त अनुरोध पाथ की जाँच इसके बजाय steady-state परीक्षण करते हैं। वे अपरिवर्तित रिलीज़ बाइनरी चलाते हैं, जिसकी पहचान उसका SHA-256 है। इस होस्ट पर रैंकिंग किसी दूसरे पर रैंकिंग नहीं है।

8.2 प्रिमिटिव लेटेंसी

तालिका 5 एक Ryzen 9 5950X कोर पर प्रति ऑपरेशन नैनोसेकंड में Sibuna के प्रिमिटिव समय-माप: सात बैच की माध्यिका, सबसे तेज़ और सबसे धीमे बैच के साथ (`benchmarks/results/primitives-latest.md`, 2026-10-15T22:34:59Z पर बनी)

सबसिस्टम	वर्कलोड	माध्यिका	न्यूनतम-अधिकतम
कार्य का प्रमाण	Hashcash सत्यापन, 16 बिट	70.2	69.9-71.0
कार्य का प्रमाण	PoSW सत्यापन, गहराई 13, 16 ओपनिंग	22,272.4	22,257.0-22,316.4
बॉट मैचर	Aho-Corasick, 40 सिग्नेचर	55.7	55.3-57.2
बॉट मैचर	अनुक्रमिक सबस्ट्रिंग खोज, 40 सिग्नेचर (संदर्भ के लिए)	763.0	762.2-763.5
IP वर्गीकारक	IPv4 CIDR लुकअप	32.0	31.8-32.2

[22] SID 0007, review history (Rust port, consolidation; daemon integration), जो SID 0014 के JA4 फ़ील्ड से पहले 719,406 बाइट दर्ज करती है। 1.0.0-beta.3 का आकार उस मॉड्यूल का है जिसे डेमन की बिल्ड स्क्रिप्ट की तरह बिल्ड किया गया (`cargo rustc --lib -p sibuna-console-ui --target wasm32-unknown-unknown --profile wasm -- -C link-arg=--compress-relocations`), और `gzip -9` से संपीड़ित किया गया।

[23] `benchmarks/results/README.md`, "Method" और "Provenance and limits" (rustc 1.99.0; डेमन `cargo build --release -p sibuna` है, 49,113,856 बाइट); पुस्तक का भाग VIII, "कार्यप्रणाली"।

सबसिस्टम	वर्कलोड	माध्यिका	न्यूनतम-अधिकतम
IP वर्गीकारक	IPv6 CIDR लुकअप	79.3	78.5-79.6
टोकन	कुंजी-युक्त BLAKE3 सत्र टोकन	155.8	155.3-156.4
टोकन	Ed25519 सत्र टोकन	40,587.5	40,544.6-40,631.1
spent set	Robin Hood spend और लुकअप	21.0	20.8-21.3
रेट लिमिटर	GCRA, हैश किया गया क्लाइंट	6.2	6.1-6.3
रेट लिमिटर	GCRA, चार नियम scope, हैश किया गया क्लाइंट	6.3	6.2-6.4
रेट लिमिटर	GCRA, बाइट के रूप में क्लाइंट (कुंजी-युक्त हैश सहित)	12.8	12.7-12.8
रेट लिमिटर	GCRA, चार नियम scope, बाइट के रूप में क्लाइंट	13.1	13.0-13.2
HTTP पार्सर	zero-copy अनुरोध और कुकी	132.5	132.3-133.2
नीति इंजन	पूरा वर्गीकरण, Shield प्रोफ़ाइल	790.8	787.2-793.1
नीति इंजन	वर्गीकरण, Gate प्रोफ़ाइल	75.9	75.6-76.1
WAF	8 KiB बॉडी, सिमैंटिक स्कैन	12,784.3	12,771.5-12,809.2
WAF	8 KiB बॉडी, SQL injection ऑडिट मोड में	15,485.4	15,460.9-15,530.1

फैलाव संकरे हैं: किसी भी पंक्ति का कोई बैच अपनी पंक्ति की माध्यिका से 4.2% से अधिक दूर नहीं है[24]। डेमन हैश किए गए क्लाइंट वाली पंक्तियों का उपयोग करता है, क्योंकि वह प्रति अनुरोध क्लाइंट का एक बार हैश करता है; कच्चे बाइट देने वाली पंक्तियों में हर जाँच पर वह कुंजी-युक्त हैश शामिल है। टाइमर पूरे बैच को घेरते हैं, कभी अकेले ऑपरेशन को नहीं, इसलिए तालिका कोई प्रति-ऑपरेशन पर्सेंटाइल रिपोर्ट नहीं करती।

8.3 पूरे उत्पाद

तालिका 6 `wrk` के तहत Sibuna: प्रति सेकंड अनुरोध, माइक्रोसेकंड में लेटेंसी, माइक्रोसेकंड में प्रति अनुरोध CPU समय और पीक resident मेमोरी; तीन रन की माध्यिकाएँ (`benchmarks/results/product-latest.md`, 2026-10-15T23:15:24Z पर बनी)

मोड	सतह	वर्कलोड (स्टेटस)	req/s	p50	p99	CPU	पीक 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
रिवर्स प्रॉक्सी	Gate	`admitted` (200)	142,498	427	768	27.68	15,896
रिवर्स प्रॉक्सी	Gate	`challenged` (200)	199,465	160	324	12.98	16,160
रिवर्स प्रॉक्सी	Gate	`allowed_static` (200)	147,303	411	759	26.88	16,324
रिवर्स प्रॉक्सी	Gate	`attack` (200)	143,900	422	745	27.67	16,348
रिवर्स प्रॉक्सी	Shield	`admitted` (200)	138,965	438	857	28.57	16,064
रिवर्स प्रॉक्सी	Shield	`challenged` (200)	195,188	162	328	13.73	16,328
रिवर्स प्रॉक्सी	Shield	`allowed_static` (200)	145,673	415	746	27.27	16,328
रिवर्स प्रॉक्सी	Shield	`attack` (403)	209,507	151	306	12.17	16,328

forward-auth मोड में इनग्रेस Sibuna से पूछता है कि किसी अनुरोध को प्रवेश दे या नहीं, और कोई ऑरिजिन नहीं होता; हर वर्कलोड प्रति सेकंड 197,532 और 211,636 अनुरोधों के बीच है, 305 से 327 us के p99 के साथ। वहाँ चार डेमन CPU संतृप्त नहीं हैं (उनमें से 2.47 से 2.72 व्यस्त हैं), जो लोड जनरेटर को ऊपरी सीमा के रूप में दिखाता है। रिवर्स-प्रॉक्सी मोड में स्वीकृत और स्टैटिक पंक्तियों में स्टब ऑरिजिन और एक अतिरिक्त loopback hop शामिल है, और डेमन के CPU संतृप्त हैं (चार में से 3.95 से 3.98 व्यस्त); जिन अनुरोधों का उत्तर Sibuna स्वयं देता है (चैलेंज, और Shield द्वारा attack की अस्वीकृति) वे अब भी प्रति सेकंड लगभग 195,000 से 210,000 पर चलते हैं। पीक resident मेमोरी हर वर्कलोड में 15,896 और 16,348 KiB के बीच रहती है[25]।

तीन रिवर्स-प्रॉक्सी पंक्तियों में लगभग बीस लाख में से कुछ उत्तर `503` थे: बंद हो रहे कनेक्शन के प्रति-पता कनेक्शन सीमा में अपना स्थान लौटाने से पहले ही `wrk` दोबारा जुड़ गया। अब डेमन समापन रिस्पॉन्स लिखने से पहले स्थान लौटा देता है; ये फ़ाइलें उस सुधार से पहले की हैं।

रिलीज़ प्रोफ़ाइल में बाद के एक बदलाव ने bundled SQLite की debug जानकारी हटा दी है, जो बिल्ड manifest की टिप्पणी के अनुसार 49 MB

[24] पुस्तक का भाग VIII, "प्रिमिटिव लेटेंसी", `primitives-latest.json` से गणना।

[25] `benchmarks/results/product-latest.md`; व्यस्त CPU `product-latest.json` के `cores_busy_median` से; `503` उत्तर और उनका सुधार `benchmarks/results/README.md` और SID 0002, revision note (bug fix) से।

तालिका 7 Sibuna डेमन का संसाधन उपयोग (`product-latest.json` और `primitives-latest.json`, 15 अक्टूबर 2026, जैसा पुस्तक के भाग VIII में तालिकाबद्ध है)

माप	मान
डेमन बाइनरी, SQLite और कंसोल वाला डिफ़ॉल्ट बिल्ड	49,113,856 bytes
निष्क्रिय resident मेमोरी, उत्पाद रन	5,592 से 5,788 KiB
किसी वर्कलोड के दौरान पीक resident मेमोरी	15,896 से 16,348 KiB
वर्कलोड के बाद resident मेमोरी	16,176 से 16,348 KiB
नीति इंजन टेबल	209,052 बाइट
बॉट सिग्नेचर टेबल	42,244 बाइट
WAF सिग्नेचर टेबल	164,908 बाइट
ब्राउज़र प्रूवर (WebAssembly)	14,744 bytes

तालिका 8 Sibuna, Shield, निश्चित प्रस्तावित दरों पर स्वीकृत अनुरोध; माइक्रोसेकंड में लेटेंसी और प्रति अनुरोध CPU समय, 10-सेकंड के तीन रन की माध्यिकाएँ (`benchmarks/results/latency-latest.md`, 2026-10-15T23:07:14Z पर बनी)

मोड	प्रस्तावित req/s	प्राप्त req/s	p50	p90	p99	p99.9	त्रुटियाँ	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
रिवर्स प्रॉक्सी	10,000	10,000	56	61	65	76	0	32.25
रिवर्स प्रॉक्सी	25,000	25,000	51	54	58	72	0	28.63
रिवर्स प्रॉक्सी	50,000	50,000	47	50	60	74	0	27.45
रिवर्स प्रॉक्सी	100,000	99,996	55	75	100	150	0	28.42
रिवर्स प्रॉक्सी	150,000	134,344	583,934	1,544,160	1,906,273	1,967,134	4	29.22

में से लगभग 13 MB है; परिणाम फ़ाइलें तब से दोबारा नहीं बनाई गईं, इसलिए हम मापा गया आकार उद्धृत करते हैं[26]।

8.4 समान प्रस्तावित लोड पर लेटेंसी

`wrk` 64 अनुरोध एक साथ चलते रखता है, इसलिए अनुरोध दर डेमन पर निर्भर करती है, और `wrk` के पर्सेंटाइल उन अनुरोधों को छोड़ देते हैं जो उसने प्रतीक्षा के दौरान नहीं भेजे। इसके बजाय open-loop हार्नेस प्रस्तावित दर तय करता है और हर देरी को डेमन के खाते में डालता है, जो किसी आगंतुक का वास्तविक अनुभव है।

forward-auth मोड में p99.9 प्रति सेकंड 150,000 अनुरोधों तक 75 us से नीचे रहता है। रिवर्स-प्रॉक्सी मोड में, पाथ में ऑरिजिन के साथ, प्रति सेकंड 100,000 अनुरोधों पर p99 100 us और p99.9 150 us है। रिवर्स-प्रॉक्सी मोड में प्रति सेकंड 150,000 अनुरोधों पर डेमन अपनी क्षमता से आगे है: वह प्रति सेकंड 134,344 अनुरोध पूरे करता है, जो उसी वर्कलोड के लिए `wrk` द्वारा मापे गए 138,965 के पास है, और लेटेंसी सेकंडों तक बढ़ जाती है, जैसा कि क्षमता से ऊपर open-loop परीक्षण को दिखाना चाहिए। उस पंक्ति की चार त्रुटियाँ ओवरलोड में हुईं[27]।

8.5 मापन ने क्या बदला

लेटेंसी हार्नेस ने कनेक्शन थ्रेड का डिज़ाइन बदल दिया। उसके पहले रन में, 64 कनेक्शन के साथ, जिनमें से हर एक को डेमन 4,096 अनुरोधों के बाद बंद करता था, प्रति सेकंड 100,000 स्वीकृत अनुरोधों पर 99.9वाँ पर्सेंटाइल 0.3 से 2 ms था, एक ऐसी टेल जो `wrk` ने नहीं दिखाई थी। थ्रेड बनाना और समाप्त करना उसके signal stack को map और unmap करता है और उसके स्टैक पेज लौटाता है, और address space के इन बदलावों ने दूसरे थ्रेड को अटका दिया। एक प्रयोग ने निदान तय कर दिया: एक बिल्ड जो कभी कनेक्शन बंद नहीं करता था, और इसलिए रन के दौरान कोई थ्रेड नहीं बनाता था, उसने उसी दर पर 54 से 60 us दिए। सुधार अपना कनेक्शन पूरा कर चुके थ्रेड को रुककर अगला कनेक्शन लेने देता है; 10 सेकंड तक रुका रहा थ्रेड समाप्त हो जाता है, इसलिए कोई बर्स्ट अपने स्टैक मेमोरी में नहीं छोड़ता। सुधार के बाद प्रति सेकंड 100,000 अनुरोधों पर p99.9 forward-auth मोड में 62 us और रिवर्स-प्रॉक्सी मोड में 150 us है[28]।

एक दुर्लभ, आवर्ती रुकावट p99 को हिलाने से बहुत पहले दूर की टेल को हिला देती है। 64 कनेक्शन पर प्रति सेकंड 100,000 अनुरोधों पर हर कनेक्शन $4096/1562.5 \approx 2.6$ s चलता है, इसलिए हर सेकंड लगभग 24 थ्रेड शुरू और समाप्त होते हैं; दस लाख अनुरोधों के रन में ऊपरी 0.1% केवल 1,000 अनुरोध है, और प्रति thread turnover कुछ विलंबित अनुरोध उसे भरने के लिए पर्याप्त हैं। ऐसे दोष को देखने के लिए p99.9 और हर अनुरोध पर टाइमर चाहिए।

8.6 मुख्य निष्कर्ष

1. एक माइक्रोसेकंड से कम में वर्गीकरण। Gate प्रोफ़ाइल में नीति इंजन ब्राउज़र अनुरोध का वर्गीकरण 75.9 ns में करता है; Shield प्रोफ़ाइल में पूरे निरीक्षण के साथ इसमें 790.8 ns लगते हैं।

[26] पुस्तक का भाग VIII, "संसाधन उपयोग (Footprint)"; रूट `Cargo.toml`, प्रोफ़ाइल `[profile.release.package.libsqlite3-sys]`।
[27] `benchmarks/results/latency-latest.md`; प्रति सेकंड 25,000 अनुरोधों वाली पंक्तियाँ और 150,000 पर रिवर्स-प्रॉक्सी पंक्तियाँ उसी फ़ाइल में हैं।
[28] SID 0002, revision note (performance, connection threads reused); पुस्तक का भाग VIII, "मापन ने क्या बदला"; `benchmarks/results/latency-latest.md`।

तालिका 9 सुविधाओं की तुलना (पुस्तक का भाग II; अन्य उत्पादों के बारे में तथ्य सितंबर 2026 में सार्वजनिक दस्तावेज़ों से पढ़े गए, JA4 वाली पंक्ति अक्टूबर 2026 में)

सुविधा	Sibuna 1.0.0-beta.3	Anubis 1.27	SafeLine CE 9	Cloudflare WAF
किस रूप में चलता है	एक बाइनरी; रिवर्स प्रॉक्सी या forward auth; वैकल्पिक SQLite फ़ाइल या साझा PostgreSQL डेटाबेस	एक Go बाइनरी; रिवर्स प्रॉक्सी या forward auth	सात Docker कंटेनर (tengine, detector, mgt, luigi, fvm, chaos, PostgreSQL)	होस्टेड नेटवर्क; DNS, Cloudflare की ओर इंगित करता है
रनटाइम	नेटिव कोड, कोई garbage collector नहीं; स्टोरेज बंद होने पर कोई तृतीय-पक्ष निर्भरता नहीं	Go, garbage collected	C/C++ प्रॉक्सी और डिटेक्टर, Go प्रबंधन, PostgreSQL	मालिकाना edge सॉफ़्टवेयर
लाइसेंस	LGPL-3.0-or-later	MIT	GPL-3.0 प्रबंधन कोड; बंद डिटेक्टर इमेज	मालिकाना सेवा
Proof-of-work प्रवेश	Hashcash (बिट-स्तर) और Cohen-Pietrzak अनुक्रमिक कार्य, प्रति नियम चुना गया	हेक्स nibble में SHA-256 Hashcash; कार्य-रहित चैलेंज	JavaScript चैलेंज और CAPTCHA; कार्य से बँधा नहीं	Managed challenge, JS challenge, Turnstile
सर्वर पर चैलेंज की अवस्था	हल होने तक कुछ नहीं; हल किए गए टैग एक निश्चित Robin Hood टेबल में	स्टोर बैकएंड: memory, bbolt, Valkey या S3	स्टैक द्वारा प्रबंधित	Cloudflare द्वारा प्रबंधित
सत्र टोकन	कुंजी-युक्त BLAKE3, 75 अक्षर, चुकाए गए कार्य से बँधा; Ed25519 वैकल्पिक	Ed25519 से हस्ताक्षरित JSON Web Token; HS512 वैकल्पिक	स्टैक द्वारा जारी कुकी	`cf_clearance` कुकी
पेलोड निरीक्षण	Shield: टैग किया गया ऑटोमेटन और संरचनात्मक टोकनाइज़र, बॉडी के पहले 8 KiB	कोई नहीं	OWASP श्रेणियों पर सिमैंटिक विश्लेषण	Managed rulesets (Pro और ऊपर)
TLS फ़िंगरप्रिंट (JA4)	नियम उस JA4 मान से मिलान करते हैं जिसे कोई विश्वसनीय इनग्रेस फ़ॉरवर्ड करता है; Sibuna उसकी गणना नहीं करता	दस्तावेज़ों में नहीं; नीतियाँ HTTP फ़िंगरप्रिंट JA4H का संदर्भ दे सकती हैं, जिसकी गणना Anubis करता है	अपने ही TLS टर्मिनेशन पर गणना; कस्टम नियम और एक बिल्ट-इन deny सूची	कस्टम नियमों और analytics में JA3 और JA4 (Bot Management के साथ Enterprise)
साझा अवस्था	नीति, प्रतिष्ठा और फ़ॉरेंसिक के लिए एक PostgreSQL डेटाबेस	साझा signing कुंजी; साझा Valkey स्टोर	प्रति होस्ट एक स्टैक	वैश्विक anycast नेटवर्क
TLS टर्मिनेशन	नहीं (इनग्रेस)	नहीं (इनग्रेस)	हाँ	हाँ

2. सममित सत्यापन। कुंजी-युक्त BLAKE3 सत्र टोकन का सत्यापन 155.8 ns में होता है, जबकि Ed25519 टोकन का 40,587.5 ns में: एक विश्वास क्षेत्र के भीतर समान अखंडता के लिए लगभग 260 गुना कम कार्य।
3. छोटा, स्थिर संसाधन उपयोग। चार CPU पर प्रति सेकंड लगभग 200,000 forward-auth अनुरोध, हर वर्कलोड में लगभग 16 MiB की पीक resident मेमोरी और निष्क्रिय अवस्था में 6 MiB से कम।
4. टेल के लिए *open-loop* मापन चाहिए। thread-turnover रुकावट closed-loop हार्नेस को दिखाई नहीं दी और open-loop जनरेटर के तहत p99.9 पर साफ़ दिखी।

9 बाज़ार से तुलना: Sibuna और उसके पड़ोसी

हम Sibuna की तुलना उन उपकरणों से करते हैं जिन्हें कोई ऑपरेटर सबसे अधिक उसके विरुद्ध तौलेगा। अन्य उत्पादों के बारे में तथ्य सितंबर 2026 में उनके सार्वजनिक दस्तावेज़ों और रिलीज़ सामग्री से पढ़े गए, और Sibuna पुस्तक उन्हें उनके स्रोतों के साथ भाग II में दर्ज करती है[31]। इनमें से हर उत्पाद अपने उपयोगकर्ताओं की अच्छी सेवा करता है, और हर एक ने ऐसे डिज़ाइन चुनाव किए हैं जिनके लाभ हमारा हार्नेस नहीं मापता। हम आर्किटेक्चर के अंतरों का वर्णन करते हैं, कोई रैंकिंग नहीं।

9.1 Anubis के साथ मापी गई तुलना

हमने Anubis v1.27.0 की आधिकारिक रिलीज़ बाइनरी को पूरे उत्पाद के हार्नेस से Sibuna के साथ मापा: वही चार वर्कलोड, हर उत्पाद के हर थ्रेड के लिए वही चार CPU, वही `wrk` लोड, बराबर Hashcash कार्य (Sibuna के लिए 8 शून्य बिट, Anubis के लिए 2 शून्य हेक्स अंक), हर उत्पाद का चैलेंज ब्राउज़र की तरह हल करके पाया गया सत्र, और ऑरिजिन के रूप में `caddy respond`[29]। यहाँ Sibuna अपनी Shield सतह चलाता है, इसलिए वह हर पेलोड का निरीक्षण भी करता है, जो Anubis नहीं करता।

इन पंक्तियों को पढ़ना. ये संख्याएँ एक मशीन पर चार वर्कलोड का वर्णन करती हैं, हर डिप्लॉयमेंट का नहीं, और कई पंक्तियाँ एक जैसी स्थितियों की तुलना नहीं करतीं। forward-auth मोड में Anubis `/robots.txt` और attack पाथ का उत्तर स्वयं देता है; Anubis किसी पेलोड का निरीक्षण नहीं करता, इसलिए Shield की अस्वीकृति अधिक कार्य करती है; चैलेंज बूटस्ट्रैप Sibuna के लिए दो अनुरोध है और Anubis के लिए एक पेज; और सत्र-जाँच के अंतर का बड़ा भाग सममित टोकन से समझ में आता है, क्योंकि Anubis की अपनी HS512 योजना उसके Ed25519 डिफ़ॉल्ट से 55% तेज़ है। Anubis एक व्यापक रूप से उपयोग होने वाला और सक्षम उपकरण है, और उसके maintainer ने अच्छे कारणों से अलग चुनाव किए हैं। मापन जो दिखाता है वह यह है कि एक छोटा, सीमित अनुरोध पाथ रक्षक के लिए प्रवेश को सस्ता बना सकता है।

[31] पुस्तक का भाग II, "सुविधाओं की तुलना" और "इंटरस्टिशियल और edge"; Sibuna का लाइसेंस `Cargo.toml` में वर्कस्पेस का `license` है।

[29] `benchmarks/results/comparison-latest.md` और `admission-latest.md`, 16 अक्टूबर 2026 को `cargo xtask bench product --anubis` और `cargo xtask bench admission --anubis` से दर्ज; पुस्तक का भाग VIII, "Sibuna और Anubis", विधि और उत्पादों के बीच हर अंतर की सूची देता है।

तालिका 10 `wrk` के तहत Sibuna (Shield) और Anubis 1.27.0: प्रति सेकंड अनुरोध और माइक्रोसेकंड में p99 लेटेंसी, पाँच-सेकंड के तीन रन की माध्यिकाएँ (`benchmarks/results/comparison-latest.md`)। जहाँ अलग न दिखाया गया हो, स्टेटस 200 है; Shield attack को 403 से अस्वीकार करता है।

मोड	वर्कलोड	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
रिवर्स प्रॉक्सी	`admitted`	103,298	3,321	19,398	11,062
रिवर्स प्रॉक्सी	`challenged`	197,327	328	28,842	9,546
रिवर्स प्रॉक्सी	`allowed_static`	106,819	3,316	28,862	30,124
रिवर्स प्रॉक्सी	`attack`	210,483 (403)	306	23,399	14,124

तालिका 11 दो CPU पर प्रति सेकंड प्रवेश ऑपरेशन, सात बैच की माध्यिका (`benchmarks/results/admission-latest.md`)। Anubis अपनी हर टोकन योजना के साथ एक बार चलता है।

उत्पाद (टोकन)	सत्र जाँच	अप्रमाणित	बूटस्ट्रैप	प्रमाण सत्यापन	resident मेमोरी
Sibuna (कुंजी-युक्त 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

forward-auth मोड में Anubis प्रति स्वीकृत अनुरोध 106 us CPU खर्च करता है, जहाँ Sibuna 13; उसकी पीक resident मेमोरी forward-auth मोड में लगभग 32 MiB थी और रिवर्स-प्रॉक्सी मोड में बढ़कर लगभग 600 MiB हो गई, जबकि Sibuna की पूरे समय लगभग 16 MiB रही। `caddy respond` के पीछे Sibuna की प्रॉक्सी की गई पंक्तियाँ ऑरिजिन से सीमित हैं, जो लोड जनरेटर के CPU साझा करता है: पूरे उत्पाद के हल्के स्टब ऑरिजिन के साथ वही पंक्तियाँ प्रति सेकंड लगभग 139,000 अनुरोधों तक पहुँचती हैं।

9.2 आर्किटेक्चर की विस्तृत तुलना

9.2.1 SafeLine (Chaitin Technology)

SafeLine libinjection के सिमैंटिक विचार को एक प्रबंधन कंसोल वाले पूरे उत्पाद तक ले जाता है, और उसका विक्रेता 71.65% (balanced मोड) और 76.17% (strict मोड) की पहचान दर और 0.07% तथा 0.22% की false-positive दर रिपोर्ट करता है[32]। ये पहचान की गुणवत्ता के मापन हैं, जो Sibuna ने अपने इंजन के लिए प्रकाशित नहीं किए हैं।

- डिप्लॉयमेंट। community edition कम से कम एक CPU कोर, 1 GB मेमोरी और 5 GB डिस्क वाले Linux होस्ट पर सात Docker कंटेनर के रूप में आता है। वह TLS स्वयं टर्मिनेट करता है, जो Sibuna नहीं करता।
- अवस्था। वह अपना आक्रमण लॉग स्टैक के भीतर PostgreSQL में रखता है, प्रति होस्ट एक स्टैक।
- *Sibuna* कहाँ अलग है। Sibuna एक बाइनरी है जो वैकल्पिक SQLite फ़ाइल के साथ शुरू होती है, और क्लस्टर एक PostgreSQL डेटाबेस साझा करता है जिसे ऑपरेटर शायद पहले से चला रहा हो। उत्पाद रन में उसकी निष्क्रिय resident मेमोरी 5,592 से 5,788 KiB थी। हमने SafeLine को अपने होस्ट पर नहीं चलाया, जिस पर कोई Docker डेमन नहीं था, इसलिए हम प्रदर्शन की कोई तुलना नहीं करते।

9.2.2 Anubis और Coraza

proof-of-work इंटरस्टिशियल 2025 में आम हुआ, जब language model के प्रशिक्षण के लिए scraper ट्रैफ़िक ने सामान्य ऑरिजिन चलाना महँगा कर दिया, और Anubis उसका सबसे प्रसिद्ध खुला इम्प्लीमेंटेशन है। यह एक सक्षम, व्यापक रूप से उपयोग होने वाला Go रिवर्स प्रॉक्सी है: यह JavaScript पहेली देता है, Ed25519 से JSON Web Token पर हस्ताक्षर करता है, और चैलेंज की अवस्था एक स्टोर में रखता है। Coraza ModSecurity की नियम भाषा (SecLang) लागू करता है और OWASP Core Rule Set चलाता है; ऐसे गेट के साथ नियम-आधारित निरीक्षण जोड़ने का यह एक तरीका है।

- कार्य कहाँ जाता है। Anubis और Sibuna Gate दोनों ऑरिजिन के कुछ भी करने से पहले क्लाइंट से कीमत चुकवाते हैं। Sibuna उन रूट के लिए, जिन्हें अधिक मज़बूत गेट चाहिए, प्रति नियम चुना गया अनुक्रमिक कार्य का दूसरा टियर जोड़ता है, और उसके सत्र याद रखते हैं कि कितना कार्य चुकाया गया।
- सर्वर क्या याद रखता है। Anubis चैलेंज संग्रहीत करता है; Sibuna एक स्वयं-प्रमाणित रिकॉर्ड जारी करता है और केवल हल किए गए टैग एक निश्चित टेबल में याद रखता है (प्रमेय 3)। उसका डिफ़ॉल्ट टोकन सममित है, जिसका सत्यापन सस्ता है और जो क्वांटम हमलावर को केवल द्विघाती गति-वृद्धि देता है।
- निरीक्षण। Sibuna Shield प्रवेश वाले पास में ही अपना संरचनात्मक निरीक्षण करता है, इसलिए एक प्रोसेस और एक पार्सर दोनों भूमिकाएँ निभाते हैं। वह OWASP Core Rule Set नहीं चलाता। हमने मौजूदा SecLang इंजनों का, libmodsecurity और Coraza के bindings सहित, मूल्यांकन किया और किसी को नहीं अपनाया: edge सर्वर पर निरीक्षण इंजन सुरक्षा के लिए निर्णायक कोड है, जिसे हम स्वयं रखना और जाँचना पसंद करते हैं। यदि Sibuna Core Rule Set के साथ संगतता जोड़ता है, तो पहले एक डिज़ाइन रिकॉर्ड उसका उपसमुच्चय निर्दिष्ट करेगा[33]।
- मापन। एक होस्ट पर समान लोड में Sibuna ने forward-auth मोड में पाँच गुना से अधिक स्वीकृत अनुरोधों का उत्तर दिया, कम p99 के साथ और प्रति अनुरोध लगभग आठवें हिस्से जितने CPU के साथ (Anubis के साथ मापी गई तुलना)।

[32]पुस्तक का भाग VIII, "जो मापा नहीं गया": विक्रेता द्वारा रिपोर्ट किए गए आँकड़े, सितंबर 2026 में जाँचे गए।
[33]SID 0010, "The OWASP Core Rule Set"।

9.2.3 क्लाउड edge WAF (Cloudflare और अन्य)

होस्टेड edge निरीक्षण, रेट लिमिटिंग और बॉट स्कोरिंग को एक वैश्विक control plane के साथ कई points of presence में रखते हैं। वे ऐसे वॉल्यूमेट्रिक हमले सोख लेते हैं जिन्हें कोई अकेला सर्वर नहीं सोख सकता, और उन्हें ऑपरेटर से कोई हार्डवेयर नहीं चाहिए। Sibuna का क्लस्टर डिप्लॉयमेंट उनका आकार उधार लेता है, स्थानीय अपरिवर्तनीय स्नैपशॉट को भरने वाला एक साझा नीति plane, और फिर भी self-hosted रहता है।

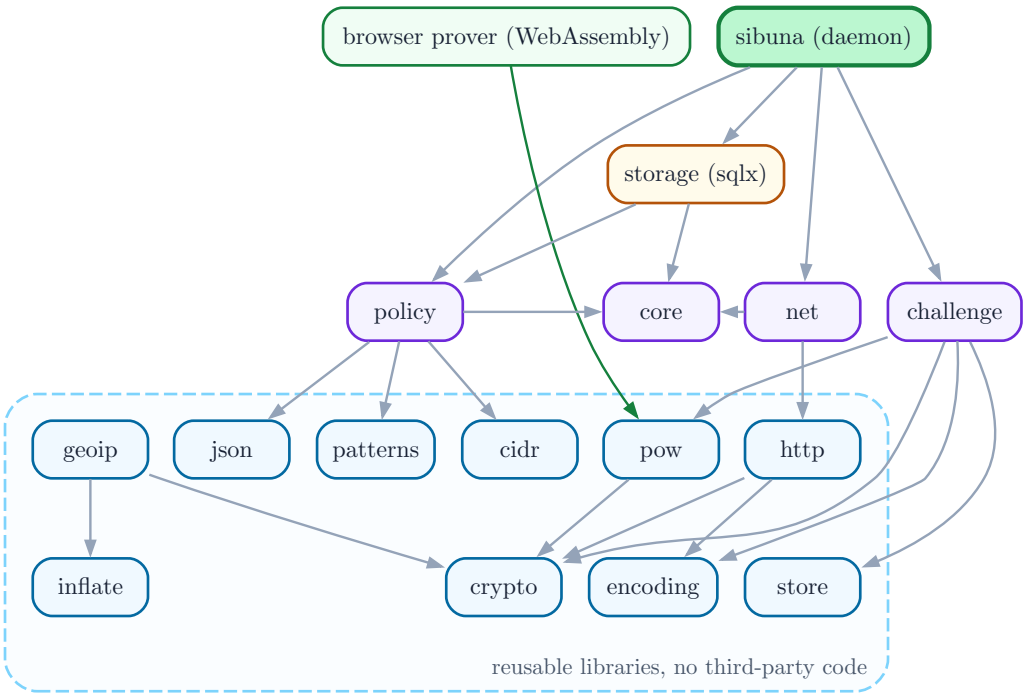
- डेटा का निवास *(data residency)*। होस्टेड WAF ऑपरेटर का TLS टर्मिनेट करता है और साझा इन्फ़्रास्ट्रक्चर पर अनुरोध पेलोड देखता है, जो डेटा-निवास की आवश्यकताओं से टकरा सकता है।
- प्लान और कोटा। सुविधाएँ प्लान से बँधी हैं: Cloudflare के प्रकाशित कोटा Free, Pro, Business और Enterprise प्लान पर 5, 20, 100 और 1,000 कस्टम नियम और 1, 2, 5 और 100 rate-limiting नियम देते हैं।
- *Sibuna* कहाँ अलग है। Sibuna ऑपरेटर की अपनी मशीनों पर बिना प्रति-अनुरोध शुल्क के चलता है, और उसकी नीति के बदलाव अगले स्टोरेज tick (डिफ़ॉल्ट 500 ms) पर हर नोड तक पहुँचते हैं। वह वॉल्यूमेट्रिक नेटवर्क हमलों से सुरक्षा नहीं देता, प्रशिक्षित मॉडल से बॉट का स्कोर नहीं निकालता, और TLS टर्मिनेट नहीं करता; जिस डिप्लॉयमेंट को ये चाहिए, उसे अब भी अपने सामने एक इनग्रेस या edge नेटवर्क चाहिए।

10 इम्प्लीमेंटेशन: छोटे क्रेट और कम निर्भरताएँ

Sibuna Rust में लिखा गया है, छोटे क्रेट के एक वर्कस्पेस के रूप में, जिनकी निर्भरताएँ एक ही दिशा में जाती हैं: दोबारा उपयोग योग्य लाइब्रेरी, फिर कॉन्फ़िगरेशन और निदान का core, फिर एप्लिकेशन लाइब्रेरी, फिर डेमन और ब्राउज़र प्रूवर। तीन इंजीनियरिंग चुनाव ऑपरेटर के लिए मायने रखते हैं, इसलिए हम उन्हें संक्षेप में दर्ज करते हैं[34]।

कम निर्भरताएँ। हम हर तृतीय-पक्ष लाइब्रेरी को एक दायित्व मानते हैं: उसकी समीक्षा करनी होती है, उसे अपडेट करना होता है और उस पर भरोसा करना होता है। data plane किसी का उपयोग नहीं करता। हैशिंग, हस्ताक्षर, base64, JSON, DEFLATE, Aho-Corasick और HTTP/1.1 पार्सर हमारी अपनी छोटी लाइब्रेरी हैं, जिन्हें दूसरे प्रोजेक्ट स्वतंत्र रूप से उपयोग कर सकते हैं। कोई निर्भरता तभी स्वीकार होती है जब वह परिपक्व, स्व-निहित हो और सचमुच कोई बड़ी समस्या हल करती हो, और उसका कारण बिल्ड manifest में उसके पास और एक डिज़ाइन रिकॉर्ड में लिखा जाता है। दो स्वीकार हुई हैं, हर एक एक ही लाइब्रेरी तक सीमित: SQLite और PostgreSQL स्टोरेज के लिए `sqlx`, जो लगभग सौ transitive क्रेट लाता है[35], और कंसोल के बाहर जाने वाले TLS तथा PostgreSQL के TLS के लिए `rustls`, रन टाइम पर तेरह क्रेट, क्योंकि अपना लिखा TLS क्लाइंट सुरक्षा के लिए निर्णायक एक ऐसा प्रोजेक्ट होता जो Sibuna के data plane से बड़ा होता[36]। स्टोरेज और कंसोल के बिना बिल्ड में कोई तृतीय-पक्ष निर्भरता नहीं होती।

मेमोरी सुरक्षा। कंपाइलर की मेमोरी-सुरक्षा जाँचों को बायपास करने वाला कोड डिफ़ॉल्ट रूप से वर्जित है, और जिस मॉड्यूल को इसकी ज़रूरत हो उसे नाम लेकर opt in करना होता है और हर उपयोग का औचित्य एक टिप्पणी में देना होता है। कुछ मॉड्यूल ऐसा करते हैं: प्रोसेसर के SHA निर्देशों वाले पाथ, सीक्रेट मिटाने वाला एक राइट, signal handling के लिए कुछ C library फ़ंक्शन की घोषणाएँ, दोनों WebAssembly मॉड्यूल के एक्सपोर्ट, और घटनाओं को स्टोरेज थ्रेड तक ले जाने वाली रिंग। रिंग ही एकमात्र स्थान है जहाँ हमने ऐसा कोड गति के लिए रखा, और वह भी एक मापन के बाद: 16



चित्र 6 Sibuna की लाइब्रेरी, सरल रूप में (SID 0010 के आधार पर)। तीर किसी लाइब्रेरी से उसकी ओर जाते हैं जिसका वह उपयोग करती है। दोबारा उपयोग योग्य लाइब्रेरी किसी एप्लिकेशन लाइब्रेरी पर निर्भर नहीं करतीं, और केवल स्टोरेज और कंसोल की लाइब्रेरी तृतीय-पक्ष कोड लाती हैं।

[34] SID 0010, "Workspace", "Dependency policy" और "Engineering standards"।
[35] SID 0011, "Why sqlx"; `Cargo.toml`, जहाँ हर तृतीय-पक्ष क्रेट अपना कारण साथ रखता है।
[36] SID 0012, "Specification", "Rationale and alternatives" और review history (PostgreSQL TLS built)।

producer के साथ standard library का bounded channel प्रति push 20 से 45% अधिक लागत लेता था[37]। बाकी जगह ओनरशिप वह कार्य करती है जो अन्यथा हाथ से लिखा मेमोरी प्रबंधन करता: पार्स किया गया अनुरोध अपने बफ़र से अधिक नहीं जी सकता, इंजन अपने अंतिम पाठक के साथ मुक्त होता है, और जब तक कोई थ्रेड किसी सॉकेट तक पहुँच सकता है, वह बंद नहीं हो सकता।

अनुशासन और परीक्षण। फ़ंक्शन की बॉडी में अधिकतम 70 तार्किक पंक्तियाँ और फ़ाइल में अधिकतम 1,408 पंक्तियाँ होती हैं, जिसे एक checker लागू करता है। लाइब्रेरी कोड संरचित त्रुटियाँ लौटाता है और कभी कुछ नहीं छापता, और उपयोगकर्ता को दिखने वाली हर त्रुटि अपना स्थान बताती है, एक अंश दिखाती है और एक संकेत देती है। 1.0.0-beta.3 का वर्कस्पेस सूट SQLite पर लगभग 2,200 परीक्षण और PostgreSQL के विरुद्ध लगभग 700 और परीक्षण चलाता है; कंसोल के इंटरफ़ेस में golden fixtures सहित 298 परीक्षण हैं; और ब्राउज़र जाँचें headless Chromium को इंटरस्टिशियल और कंसोल के हर पेज से गुज़ारती हैं[30]।

11 सीमाएँ और खुला कार्य

हम सूची देते हैं कि यह पत्र क्या स्थापित नहीं करता, ताकि कोई दावा अपने साक्ष्य से अधिक व्यापक रूप में न पढ़ा जाए।

- केवल एक मापा गया पड़ोसी। हमने Anubis को एक होस्ट पर मापा (Anubis के साथ मापी गई तुलना)। हमने न SafeLine को मापा है, न किसी होस्टेड WAF को।
- लोड में कोई क्लस्टर नहीं। एक कार्यात्मक परीक्षण एक PostgreSQL डेटाबेस पर तीन नोड चलाता है (SID 0011, committed), लेकिन कोई हार्नेस लोड में कई नोड नहीं चलाता।
- *PostgreSQL* के लिए स्रोत से बिल्ड चाहिए। रिलीज़ की बाइनरी में SQLite स्टोरेज और कंसोल हैं, लेकिन PostgreSQL समर्थन नहीं; क्लस्टर Sibuna को `postgres-tls` feature के साथ बिल्ड करता है।
- कंसोल का पृथक्करण गेट लोड में आठ सक्रिय डैशबोर्ड के साथ नहीं चलाया गया है (कंसोल)।
- केवल *loopback*। कोई मापन TLS, wide-area नेटवर्क या लोड में वास्तविक ब्राउज़र को नहीं ढकता; हर डिप्लॉयमेंट को अपना पाथ स्वयं मापना चाहिए।
- सीमित निरीक्षण। Shield बॉडी के पहले 8 KiB का ह्यूरिस्टिक निरीक्षक है, पूरा SQL या HTML पार्सर नहीं, और forward-auth निरीक्षण केवल वे बाइट देखता है जो इनग्रेस भेजता है।
- प्रोटोकॉल का दायरा। Sibuna HTTP/1.0 और HTTP/1.1 बोलता है; उसमें न HTTP/2 है, न आने वाले TLS का टर्मिनेशन।
- *TLS* फ़िंगरप्रिंट इनग्रेस से आते हैं। Sibuna स्वयं JA4 की गणना नहीं कर सकता। फ़िंगरप्रिंट उतना ही विश्वसनीय है जितना उसे लिखने वाला इनग्रेस, और उस इनग्रेस को क्लाइंट द्वारा भेजी गई हर प्रति अधिलेखित करनी होगी। कोई क्लाइंट विशेष लाइब्रेरी से ब्राउज़र के handshake की नकल कर सकता है, इसलिए JA4 स्वचालन की लागत बढ़ाता है; यह पहचान सिद्ध नहीं करता[11]।
- क्रिप्टोग्राफ़िक आश्वासन। अनुक्रमिक कार्य का टियर random-oracle model में प्रकाशित प्रमाणों पर टिका है; हमारे ठोस प्रोटोकॉल के लिए एक संदर्भ labeller के विरुद्ध परीक्षण हैं, कोई स्वतंत्र क्रिप्टोग्राफ़िक ऑडिट नहीं।
- एक मशीन। हर संख्या एक AMD Ryzen 9 5950X पर ली गई। इस होस्ट पर रैंकिंग किसी दूसरे पर रैंकिंग नहीं है।

12 निष्कर्ष

वेब सुरक्षा प्रायः परत-दर-परत जुड़ती गई है: एप्लिकेशन के सामने interpreted रनटाइम, कंटेनर, रेगुलर-एक्सप्रेशन नियम-समुच्चय और बाहरी स्टोर का ढेर। Sibuna के साथ हमने उलटी दिशा आज़माई है, कुछ सरल विचारों के मार्गदर्शन में:

- प्रेषक से प्रवेश की कीमत जाँचे जा सकने वाले कार्य के रूप में लें, सर्वर की जाँच सस्ती और स्टेटलेस रखें, और सत्र को याद रखने दें कि उसने कितना कार्य चुकाया;
- प्रवेश वाले पास में ही, रैखिक समय में, संरचना से निरीक्षण करें, ताकि कोई सत्र निरीक्षण को बायपास न कर सके;
- मशीन का सम्मान करें: प्रति कनेक्शन एक बफ़र, अनुरोध पाथ पर कोई आवंटन नहीं, कुंजी-युक्त स्थान-निर्धारण वाली सीमित टेबल, और ऐसे थ्रेड जो दोबारा बनाए जाने के बजाय दोबारा उपयोग होते हैं; और
- टिकाऊ साझा अवस्था एक परिपक्व डेटाबेस के ज़िम्मे छोड़ें, एक ऐसे स्टोरेज थ्रेड के पीछे जिसकी अनुरोध पाथ कभी प्रतीक्षा नहीं करता।

एक 16-कोर मशीन पर इन चुनावों से Sibuna चार CPU पर forward-auth मोड में प्रति सेकंड 200,000 से अधिक स्वीकृत अनुरोध सर्व करता है, उस लोड में लगभग 16 MiB resident मेमोरी के साथ, और प्रति सेकंड 100,000 अनुरोधों पर 62 us के p99.9 के साथ। ये loopback पर एक होस्ट की संख्याएँ हैं, और हम इन्हें रैंकिंग के रूप में नहीं, इस साक्ष्य के रूप में देते हैं कि डिज़ाइन एक साथ टिकता है। हमने अपना consensus प्रोटोकॉल हटा दिया और यह साफ़ कहते हैं; बदले में प्रणाली छोटी है और उसके data plane में कोई तृतीय-पक्ष निर्भरता नहीं है। बहुत कुछ मापा जाना बाकी है, और हमने ठीक-ठीक बताने का प्रयास किया है कि क्या।

13 स्रोत और पुनरुत्पादन

इस पत्र की हर संख्या Sibuna रिपॉज़िटरी की किसी कमिट की गई परिणाम फ़ाइल, किसी Sibuna Discussion (SID) रिकॉर्ड या Sibuna पुस्तक से आती है, और हर तालिका या फ़ुटनोट अपने स्रोत का नाम देता है। परिणाम फ़ाइलें `cargo xtask bench primitives`, `cargo xtask bench product` और `cargo xtask bench latency` लिखते हैं, और `benchmarks/results/README.md` हर फ़ाइल के कमांड, CPU समुच्चय और उद्गम बताता है।

[37] SID 0010, review history (security review, 2026-10-15): Vyukov रिंग के विरुद्ध standard bounded channel, `tests/timing.rs`। opt-in मॉड्यूल `crates/crypto/src/sha256/` (x86-64 और AArch64), `crates/crypto/src/ct.rs`, `cmd/sibuna/src/signals.rs`, `cmd/sibuna-pow-wasm/src/lib.rs`, `cmd/console-ui/src/abi.rs` और `crates/store/src/ring.rs` हैं; वर्कस्पेस `unsafe_code = "deny"` और Clippy का `undocumented_unsafe_blocks = "deny"` सेट करता है।

[30] SID 0010, "Acceptance criteria and implementation status" (criteria 1, 3 और 5), जिसने SID 0014 से पहले लगभग 2,100 वर्कस्पेस परीक्षण और 290 इंटरफ़ेस परीक्षण दर्ज किए; 1.0.0-beta.3 की गिनतियाँ `cargo test --workspace -- --list` से हैं।

- `benchmarks/results/primitives-latest.md`, `product-latest.md`, `latency-latest.md` और उनकी `.json` फ़ाइलें; `console-8-dashboards-2026-10-15.md`।
- SID 0002 (आधारभूत आर्किटेक्चर और उसके revision नोट), SID 0004 (Shield सतह), SID 0006 (गणितीय आधार), SID 0007 (कंसोल), SID 0010 (वर्कस्पेस, क्रेट और निर्भरता नीति), SID 0011 (sqlx के ज़रिए स्टोरेज), SID 0012 (कंसोल TLS), SID 0013 (कंसोल स्टोरेज) और SID 0014 (इनग्रेस से JA4 फ़िंगरप्रिंट), `docs/sid/records/` में।
- Sibuna पुस्तक, `docs/book/`: भाग I (एक अनुरोध की लागत), भाग II (पूर्व कार्य), भाग III (क्रिप्टोग्राफ़ी), भाग IV (प्रोटोकॉल), भाग V (शून्य-आवंटन इंजीनियरिंग), भाग VI (ऑटोमेटा और सिमैंटिक फ़ायरवॉल), भाग VIII (प्रायोगिक मूल्यांकन) और भाग IX (संचालन)।
- मूल साहित्य: Back, Hashcash (1997); Cohen और Pietrzak, Simple Proofs of Sequential Work (EUROCRYPT 2018); Blocki, Lee और Zhou, On the Security of Proofs of Sequential Work in a Post-Quantum World (ITC 2021); Aho और Corasick, Efficient String Matching (1975); Celis, Robin Hood Hashing (1986); ATM Forum का Generic Cell Rate Algorithm (1996)।