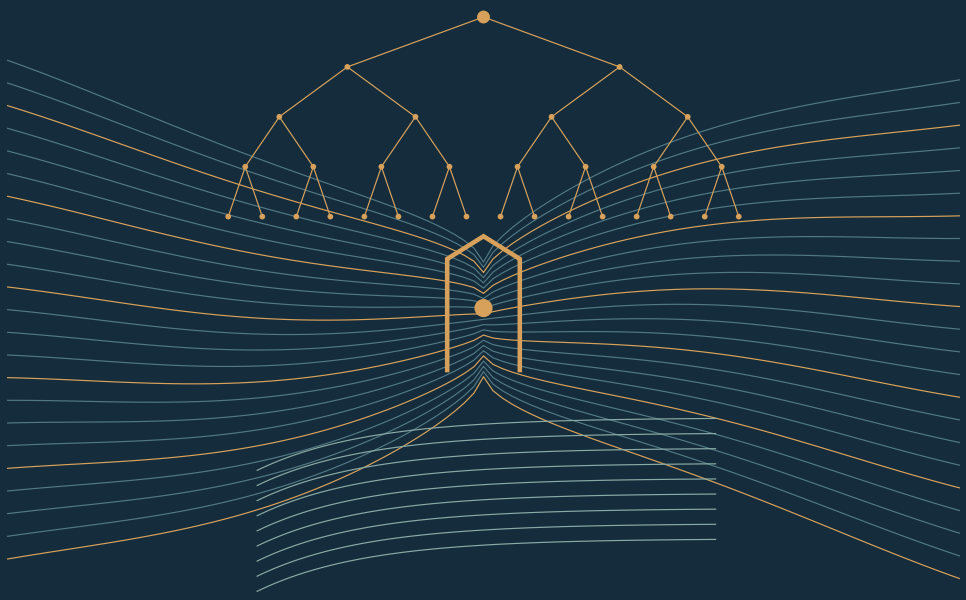
तकनीकी मार्गदर्शिका

SIBUNA

वेब सुरक्षा का गणित और इंजीनियरिंग



कार्य · विश्वास · सीमित अवस्था

Vikrant Rathore, Ronak Rathore और Kanak Rathore के सहयोग से

संस्करण 1.0.0-beta.3 · Nov 13, 2026

विषय-सूची

प्रस्तावना **6**

यह पुस्तक कैसे पढ़ें **7**
- पढ़ने के तीन रास्ते 7
- परिपाटियाँ 7

भाग I: एक अनुरोध की लागत **9**
- प्रवेश एक आर्थिक निर्णय है 9
- Proof of work क्या स्थापित करता है 9
- यादृच्छिक चर के रूप में हैश खोज 10
- सत्र कार्य की लागत बाँटते हैं 10
- सीमित अवस्था एक दूसरा बजट है 11
- वे अपरिवर्तनीय नियम जिन्हें हम आगे ले जाएँगे 11

भाग II: पूर्व कार्य और डिज़ाइन का क्षेत्र **12**
- Sibuna कहाँ से आता है 12
- एक बाइनरी में तीन विचार 14
- सुविधाओं की तुलना 15

भाग III: क्रिप्टोग्राफ़ी अनुसंधान से आधार **20**
- Hashcash: टियर एक का गणित 20
- अनुक्रमिक कार्य का प्रमाण: टियर दो 21
- सममित प्रमाणीकरण: चैलेंज और सत्र 24

भाग IV: Sibuna प्रोटोकॉल **27**
- चैलेंज का राउंड ट्रिप 27
- स्टेटलेस चैलेंज रिकॉर्ड 29
- सत्यापन का क्रम और एक-बार उपयोग 30
- सत्र कुकी 31

भाग V: Rust में शून्य-आवंटन इंजीनियरिंग **34**
- कनेक्शन लूप 34
- आवंटन के बिना समवर्ती अवस्था 44

भाग VI: ऑटोमेटा, trie और सिमैंटिक फ़ायरवॉल **54**
- एक पास में अनेक पैटर्न का मिलान 54
- IPv4 और IPv6 के लिए radix trie 56
- सिमैंटिक फ़ायरवॉल 56
- घोषणात्मक नीति इंजन 58

भाग VII: ब्राउज़र का proof-of-work इंजन **60**
- एक स्रोत, दो टारगेट 60
- वर्कर प्रोटोकॉल 63
- कैलिब्रेशन 64
- इंटरस्टिशियल 65

भाग VIII: प्रायोगिक मूल्यांकन **66**

कार्यप्रणाली 66
प्रिमिटिव लेटेंसी 70
पूरे उत्पादों की तुलना 73
समान प्रस्तावित लोड पर लेटेंसी 75
Sibuna और Anubis 77
कंसोल का अतिरिक्त भार 80
जो मापा नहीं गया 81
भाग IX: संचालन और डिप्लॉयमेंट 84
सतहें और कॉन्फ़िगरेशन 84
कंसोल प्रीव्यू 87
डिप्लॉयमेंट टोपोलॉजी 103
घोषणात्मक नीति 110
स्थायी स्टोरेज और क्लस्टरिंग 111
पैकेजिंग 115
भाग X: डेस्क संदर्भ और निदान 119
एंडपॉइंट 119
मेट्रिक 120
स्टेटस कोड 121
त्रुटि सूची 122
नीति स्कीमा 123
स्टोरेज टेबल 124
बिल्ड टारगेट 124
भाग XI: त्वरित संदर्भ कार्ड 126
बिल्ड 126
चलाना 126
कंसोल (वैकल्पिक प्रीव्यू) 126
सबसे पहले जानने योग्य फ़्लैग 127
`/__sibuna/` के अंतर्गत एंडपॉइंट 128
स्टेटस कोड 128
हेडर 129
नीति फ़ाइल का ढाँचा 129
SQL में स्टोरेज (Edge) 129
निदान 130
याद रखने योग्य संख्याएँ 130
इनग्रेस रूटिंग 130
चुनिंदा समाधान 132
भाग I: लागत और प्रायिकता 132
भाग II: पूर्व कार्य 132
भाग III: क्रिप्टोग्राफ़ी 132
भाग IV: प्रोटोकॉल 133
भाग V: अवस्था और क्रम 133
भाग VI: एल्गोरिदम 133

भाग VII: ब्राउज़र इंजन . . . 133
भाग VIII: मूल्यांकन . . . 134
भाग IX: संचालन . . . 134
भाग X: संदर्भ . . . 134
जाँच जारी रखें . . . 134
शब्दावली . . . 135
ग्रंथसूची . . . 138
अनुक्रमणिका . . . 139

© 2026, Vikrant Rathore और Sibuna के योगदानकर्ता

वेब सुरक्षा का गणित और इंजीनियरिंग।

प्रस्तावना

हर अनुरोध एक मेथड, एक पाथ, कुछ हेडर और सर्वर का ध्यान पाने का दावा लेकर आता है। इनमें से कोई भी बाइट यह स्थापित नहीं करता कि उसे सर्व करना सार्थक है। Sibuna अनुरोध और ऑरिजिन के बीच एक निर्णय रखता है: उसे प्रवेश दें, अस्वीकार करें, या उसके भेजने वाले से जाँचा जा सकने वाला कार्य करने को कहें। यह पुस्तक उस निर्णय को उसके गणितीय मॉडल से लेकर मेमोरी में रखे बाइट तक समझाती है।

सिस्टम की दो सतहें हैं। *Gate* proof-of-work आधारित प्रवेश और अनुरोधों की बाढ़ पर स्थानीय नियंत्रण देता है। *Shield* एप्लिकेशन निरीक्षण और नीति निर्णय जोड़ता है। दोनों में से कोई भी नीति, प्रतिष्ठा और घटना इतिहास के लिए स्थायी स्टोरेज का उपयोग कर सकता है, जो एक नोड पर स्थानीय हो या किसी क्लस्टर द्वारा साझा किया गया हो; इस डिप्लॉयमेंट को हम *Edge* कहते हैं। वैध सत्र प्रवेश स्थापित करता है; वह अनुरोध को निरीक्षण से छूट नहीं देता। टिकाऊ अवस्था को क्लस्टर में साझा करने से वह वितरित होती है; इससे कोई स्थानीय रेट काउंटर वैश्विक कोटा नहीं बन जाता।

इंजीनियरिंग का केंद्रीय प्रश्न यह नहीं है कि कोई हैश तेज़ है या नहीं। प्रश्न यह है कि जब कोई अविश्वसनीय क्लाइंट इनपुट चुनता है, तब भी पूरा पाथ सीमित रहता है या नहीं। एक पार्सर कितने बाइट जाँच सकता है? एक चैलेंज कितनी मेमोरी ले सकता है? एक क्लाइंट कितने कनेक्शन खुले रख सकता है? कौन-से राइट दोबारा किए जा सकते हैं? किसी नोड या डेटाबेस के गायब हो जाने के बाद क्या उपलब्ध रहता है? हर उत्तर में एक अपरिवर्तनीय नियम (invariant) और उसकी सीमा, दोनों का नाम होना चाहिए।

गणित भी इसी उद्देश्य की सेवा करता है। अपेक्षित लागत लेटेंसी की गारंटी नहीं है। किसी निश्चित कमिटमेंट के लिए सैंपलिंग का तर्क हर अनुकूली (adaptive) प्रूवर के विरुद्ध प्रमाण नहीं है। कतार एक बर्स्ट को सोख लेती है; वह स्थायी रूप से धीमे उपभोक्ता की भरपाई नहीं कर सकती। हम छोटे मॉडल निकालते हैं, उदाहरण हाथ से हल करते हैं, और फिर पूछते हैं कि इम्प्लीमेंटेशन उनसे कहाँ अलग होता है।

पुस्तक तीन तरह से उपयोग के लिए लिखी गई है। क्रम से पढ़ें तो भाग I से VII एक पाठ्यक्रम हैं: हर अध्याय यह बताकर शुरू होता है कि उसके बाद हमें क्या कर पाना चाहिए, एक उदाहरण हाथ से हल करता है, और एक अभ्यास तथा अपने शब्दों में समझाने के लिए एक अपरिवर्तनीय नियम के साथ समाप्त होता है; चुनिंदा समाधान पुस्तक के अंत में हैं। भाग VIII से X संदर्भ हैं: अपने स्रोत सहित मापन, हर फ़्लैग और टोपोलॉजी, हर एंडपॉइंट, स्टेटस, त्रुटि और तालिका। भाग XI टर्मिनल के लिए दो पन्नों का कार्ड है। पुस्तक के अंत में शब्दावली और ग्रंथसूची है।

भाग II Sibuna को सुविधा-दर-सुविधा Anubis, SafeLine और Cloudflare WAF के साथ रखकर देखता है, और भाग VIII Sibuna के अपने मापन प्रस्तुत करता है, अकेले प्रिमिटिव से लेकर पूरे उत्पाद के रन तक, और एक ही लोड जनरेटर के तहत पूरे उत्पादों के रूप में Sibuna की तुलना Anubis से करता है। दोनों इस तरह लिखे गए हैं कि पाठक दावों की जाँच कर सके: तुलना अपने स्रोतों के नाम देती है, और हर मापन उस कमिट की गई परिणाम फ़ाइल का नाम देता है जिससे वह लिया गया है।

पाठक को बुनियादी प्रोग्रामिंग, लघुगणक और सशर्त प्रायिकता का ज्ञान होना चाहिए। Rust को भाषा के सर्वेक्षण के बजाय ओनरशिप और डेटा लेआउट के ज़रिए समझाया गया है। अध्याय क्रम से पढ़े जा सकते हैं: लागत मॉडल प्रोटोकॉल की प्रेरणा देता है, प्रोटोकॉल अवस्था तय करता है, और अवस्था समवर्तिता (concurrency) तथा स्टोरेज का डिज़ाइन तय करती है। बाद के अध्याय इन अपरिवर्तनीय नियमों को मापनों और संचालन की प्रक्रियाओं में बदलते हैं।

यह इम्प्लीमेंटेशन की पुस्तक है, यह दावा नहीं कि गणना वाली पहेलियाँ स्वचालित ट्रैफ़िक को समाप्त कर देती हैं। क्लाइंट गणना खरीद सकते हैं, पते साझा हो सकते हैं, और एप्लिकेशन का सिंटैक्स किसी सीमित डिटेक्टर से कहीं समृद्ध है। उपयोगी परिणाम एक ऐसा सिस्टम है जिसकी लागतें और सीमाएँ किसी नारे के पीछे छिपाए बिना जाँची, परखी और बदली जा सकती हैं।

Sibuna 1.0.0-beta.3* के लिए संस्करण, नवंबर *2026

स्रोत, अभ्यास और दोहराए जा सकने वाले मापन Sibuna रिपॉज़िटरी के साथ उपलब्ध हैं।

यह पुस्तक कैसे पढ़ें

हम एक ही प्रश्न से शुरू करते हैं: सर्वर अभी सबसे सस्ता सुरक्षित निर्णय कौन-सा ले सकता है? प्रोटोकॉल और इम्प्लीमेंटेशन के अध्यायों में हम यह प्रश्न अपने साथ रखते हैं। गलत अनुरोध को प्रवेश देने वाला सस्ता ऑपरेशन एक दोष है; असीमित लागत वाला सही ऑपरेशन दूसरी तरह का दोष है।

प्रश्न	उत्तर किन भागों में मिलता है
पहेली से क्या मिलता है?	भाग I से III: लागत, प्रायिकता और प्रमाण की सीमाएँ।
Sibuna की तुलना कैसी है?	भाग II: परंपरा और सुविधाओं की तालिका; भाग VIII: मापन।
सत्र का क्या अर्थ है?	भाग IV: बाइंडिंग, सत्यापन का क्रम, समाप्ति और रीप्ले।
मेमोरी कहाँ जाती है?	भाग V और VI: बफ़र, थ्रेड, टेबल, ऑटोमेटा और प्रकाशन।
ब्राउज़र पहेली कैसे हल करता है?	भाग VII: वर्कर संदेश, मेमोरी और कैलिब्रेशन।
हम कैसे जानें कि यह काम करता है?	भाग VIII: तीन हार्नेस और हर एक किन बातों को बाहर रखता है।
हम इसे कैसे चलाएँ?	भाग IX: फ़्लैग, टोपोलॉजी, नीति, स्टोरेज, क्लस्टरिंग, पैकेजिंग।
वह त्रुटि क्या थी?	भाग X: एंडपॉइंट, स्टेटस, त्रुटियाँ, स्कीमा; भाग XI: कार्ड।

पढ़ने के तीन रास्ते

- डिज़ाइन सीखना। भाग I से IV क्रम से, फिर भाग VI में हल किया गया ट्रेस, फिर भाग VII। अभ्यास हल करें; पुस्तक के अंत में दिए समाधानों से तुलना उसके बाद ही करें। जब कोई ट्रेस किसी साझा टेबल या उधार ली गई स्लाइस तक पहुँचे, तो भाग V पर लौटें।
- डिप्लॉयमेंट चलाना। भाग IX, फिर भाग XI, और जानकारी खोजने के लिए भाग X खुला रखें। सतह चुनने से पहले भाग II में "सुविधाओं की तुलना" पढ़ें, और वर्कर तथा कनेक्शन सीमाएँ चुनने से पहले भाग VIII में "पूरे उत्पादों की तुलना" पढ़ें।
- कोड बदलना। भाग V और VI, हर अध्याय के स्रोत-संदर्भ, और वे SID रिकॉर्ड जिनका वे हवाला देते हैं। भाग VIII बताता है कि जो कुछ हमने बदला है उसमें रिग्रेशन कौन-सा हार्नेस पकड़ेगा; भाग II उन शर्तों की सूची देता है जिन्हें किसी बदलाव को बनाए रखना चाहिए।

परिपाटियाँ

हल किया गया उदाहरण केवल उत्तर नहीं, बीच की अवस्थाएँ भी दिखाता है। अभ्यास हमसे एक धारणा बदलने को कहता है। संकेत पहला कदम सुझाते हैं; पुस्तक के अंत में चुनिंदा समाधान तर्क को जाँचने योग्य बनाते हैं। चित्र अनुरोध पाथ को बैकग्राउंड कार्य से अलग दिखाते हैं। स्रोत-संदर्भ उस इम्प्लीमेंटेशन का नाम देता है जिसे तब देखना चाहिए जब पाठ और कोड में असहमति लगे। "इम्प्लीमेंटेशन" शीर्षक वाले बॉक्स एक फ़ंक्शन और उसकी फ़ाइल का नाम देते हैं; "अपरिवर्तनीय नियम समझाएँ" शीर्षक वाले बॉक्स हमसे विचार को वापस समझाने को कहते हैं।

समीकरणों में b बिट में Hashcash की कठिनाई है, n अनुक्रमिक कार्य की गहराई है, t ओपनिंग की संख्या है, p प्रति प्रयास सफलता की प्रायिकता है, K प्रयासों की संख्या है, T रेट लिमिटर का उत्सर्जन अंतराल (emission interval) है, τ उसका बर्स्ट टॉलरेंस है, और Q कतार की क्षमता है। जब तक अन्यथा न कहा जाए, कोई प्रतीक अपने अनुभाग तक सीमित है। भाग VIII में नैनोसेकंड और प्रति सेकंड अनुरोध किसी नामित रन के मापन हैं। हल किए गए उदाहरणों की संख्याएँ चुने हुए इनपुट हैं, बेंचमार्क के दावे नहीं। सांख्यिकीय मॉडल निष्कर्ष निकालने से पहले अपनी धारणाएँ बताते हैं।

भाग I

एक अनुरोध की लागत

कोई एल्गोरिदम चुनने से पहले उस संसाधन को पहचानें जिसकी रक्षा उसे करनी है। यह अध्याय वे छोटे मॉडल बनाता है जिनके आधार पर आगे का हर डिज़ाइन निर्णय मापा जाता है: ऑरिजिन का कार्य, पहेली का कार्य, सत्र पर लागत का बँटवारा *(amortisation)*, और वह मेमोरी जिसे कोई अविश्वसनीय क्लाइंट सर्वर से रखवा सकता है।

उद्देश्य इस अध्याय के अंत तक हमें यह कर पाना चाहिए: वह असमिका लिखना जो तय करती है कि कोई गेट अपनी लागत वसूल करता है या नहीं; बताना कि proof of work क्या स्थापित करता है और क्या नहीं; हैश खोज की अपेक्षा (expectation) और टेल निकालना; समझाना कि सत्र कार्य की लागत को कैसे बाँटते हैं; और उन पाँच अपरिवर्तनीय नियमों की सूची देना जिन्हें शेष पुस्तक बनाए रखती है।

प्रवेश एक आर्थिक निर्णय है

मान लें कि ऑरिजिन हर स्वीकृत अनुरोध के लिए डेटाबेस लुकअप करता है और एक पेज रेंडर करता है। मान लें c_o वह कार्य है, c_g प्रति अनुरोध गेट का कार्य है, और r आगमन दर है। गेट न हो तो ऑरिजिन को प्रति सेकंड rc_o इकाई कार्य संभालना होगा। जो गेट a अंश को प्रवेश देता है, वह इस माँग को rac_o कर देता है, जबकि स्वयं rc_g खर्च करता है। गेट केवल तभी उपयोगी है जब ऑरिजिन के बचाए गए कार्य की मात्रा उसकी अपनी लागत और वैध आगंतुकों पर डाली गई लागत से अधिक हो।

$$rc_g + rac_o < rc_o \quad \Leftrightarrow \quad c_g < (1-a)c_o.$$

यह क्षमता का मॉडल है, मापन नहीं। इसमें बैंडविड्थ, कनेक्शन की अवस्था, स्टोरेज और क्लाइंट की देरी शामिल नहीं हैं। इसका मूल्य यह है कि हर छूटा हुआ पद दिखाई देता है। कैश हिट c_o को छोटा कर सकता है; कोई महँगी क्वेरी उसे बड़ा कर सकती है। अनुरोधकर्ता और सर्वर की लागत का कोई सार्वभौमिक अनुपात नहीं है।

हल किया गया उदाहरण: गेट कब अपनी लागत वसूल करता है $c_o = 100$ कार्य इकाइयाँ और $c_g = 1$ चुनें। यदि आधे अनुरोध अस्वीकार किए जाते हैं, तो गेट वाला सिस्टम प्रति आगमन 100 के बजाय $1 + 0.5 \times 100 = 51$ इकाइयाँ खर्च करता है। यदि हज़ार में से केवल एक अस्वीकार होता है, तो वह 100.9 खर्च करता है: इस मॉडल में गेट जितना बचाता है उससे अधिक खर्च करता है। ये चुनी हुई संख्याएँ समय-माप नहीं हैं। वे दिखाती हैं कि हर प्रदर्शन दावे में वर्कलोड का उल्लेख क्यों होना चाहिए।

Proof of work क्या स्थापित करता है

क्लाइंट पहेली यह स्थापित करती है कि किसी ने एक सार्वजनिक सत्यापन नियम को संतुष्ट करने वाला इनपुट खोजा। वह यह स्थापित नहीं करती कि भेजने वाला मनुष्य है, न उसकी पहचान, न उसका अच्छा इरादा। अनुरोधकर्ता गणना किराये पर ले सकता है, किसी वैध सत्र को उसकी अवधि के भीतर दोबारा उपयोग कर सकता है, या कार्य को कई मशीनों में बाँट सकता है। पहेली प्रवेश की लागत बदलती है; स्वीकृत अनुरोध क्या कर सकते हैं, यह अब भी नीति और निरीक्षण तय करते हैं।

पते पर आधारित सीमाएँ पूरक हैं। वे एक पते पर अंकुश लगाती हैं, लेकिन साझा पते कई लोगों का प्रतिनिधित्व कर सकते हैं और एक अनुरोधकर्ता कई पतों का उपयोग कर सकता है। न पहेली और न ही पता पहचान बताने वाला ओरेकल है। इसलिए Sibuna प्रवेश, निरीक्षण और प्रतिष्ठा को अलग-अलग निर्णय रखता है।

यादृच्छिक चर के रूप में हैश खोज

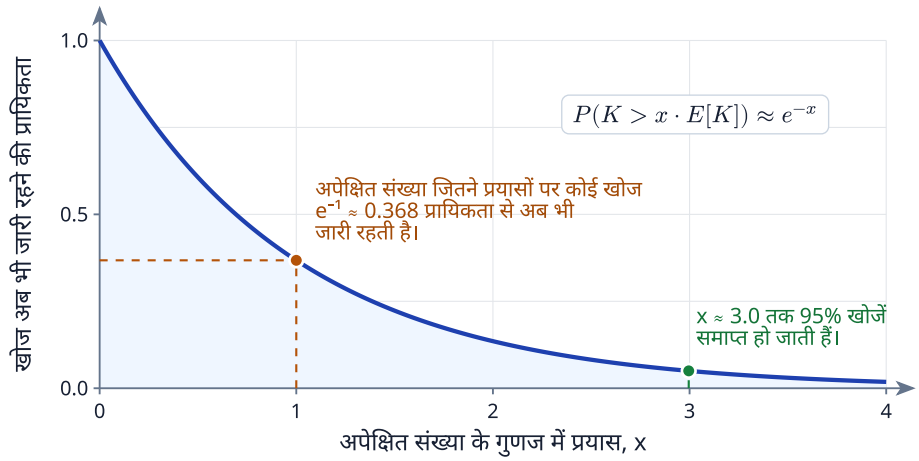
किसी आदर्श 256-बिट डाइजेस्ट में b शुरुआती शून्य बिट माँगने पर हर स्वतंत्र प्रयास की सफलता की प्रायिकता $p = 2^{-b}$ होती है। मान लें K पहली सफलता तक के प्रयासों की संख्या है। तब

$$P(K = k) = (1-p)^{k-1}p, \quad P(K > k) = (1-p)^{k}, \quad E[K] = 1/p = 2^{b}.$$

अपेक्षा टेल के योग से निकलती है:

$$E[K] = \sum_{k=0}^{\infty} P(K > k) = \sum_{k=0}^{\infty} (1-p)^{k} = 1/p.$$

कठिनाई का एक चरण अपेक्षित प्रयासों को दोगुना करता है। यह वादा नहीं करता कि हर पहेली में दोगुना समय लगेगा। कुछ पहले प्रयास में सफल होती हैं; कुछ में औसत से कहीं अधिक समय लगता है। जो क्लाइंट केवल एक हल को मापता है, उसने मुख्यतः इसी यादृच्छिकता को मापा है।



चित्र 1 हैश खोज की उत्तरजीविता प्रायिकता (survival probability): अपेक्षित प्रयास संख्या के x गुना प्रयासों के बाद भी खोज जारी रहने की प्रायिकता, बड़े कार्य वाले सन्निकटन $P(K > x/p) \approx e^{-x}$ में।

हल किया गया उदाहरण: अपेक्षा कोई समय-सीमा नहीं है $b = 16$ पर अपेक्षित कार्य $65{,}536$ हैश है। इतने प्रयासों के बाद भी खोज जारी रहने की प्रायिकता लगभग $e^{-1} = 0.368$ है। 95वें पर्सेंटाइल की प्रयास संख्या $\lceil \ln(0.05) / \ln(1 - 2^{-16}) \rceil \approx 196327$ है, जो औसत की लगभग तीन गुनी है। यूज़र इंटरफ़ेस को अपेक्षित पूर्णता समय पर विफलता घोषित करने के बजाय इस फैलाव को सहना चाहिए।

सत्र कार्य की लागत बाँटते हैं

मान लें कि एक सत्र समाप्ति से पहले m अनुरोधों की अनुमति देता है, पहेली की लागत c_p है, सत्यापन की लागत c_v है, और सत्र जाँच की लागत c_s है। असफल प्रयासों को छोड़ दें, तो प्रति अनुरोध गेट की औसत (amortized) लागत

$c_v/m + c_s$ है, जबकि क्लाइंट की पहेली लागत c_p/m है। सत्र की अवधि बढ़ाना लोगों और स्वचालित क्लाइंट दोनों की समान रूप से मदद करता है। इसे चुनना नीति का निर्णय है, क्रिप्टोग्राफ़िक अनुकूलन नहीं।

अभ्यास 1.1. एक क्लाइंट को प्रति सत्र 100 अनुरोध मिलते हैं और वह दस लाख प्रयासों वाली पहेली हल करता है। प्रति अनुरोध औसत कार्य कितना है? यदि वह सत्र को किसी दूसरी प्रोसेस के साथ साझा करे तो क्या बदलता है? संकेत: लेखांकन मॉडल और टोकन की वास्तविक बाइंडिंग में अंतर करें।

सीमित अवस्था एक दूसरा बजट है

कार्य को अनुरोध पाथ से हटाने से वह गायब नहीं हो जाता। मान लें घटनाएँ λ दर से आती हैं, स्टोरेज श्रेड उन्हें औसत दर μ से सहेजता है, और कतार Q रिकॉर्ड रख सकती है। जब $\lambda > \mu$ हो, तो एक तरल (fluid) सन्निकटन देता है

$$t_{\text{fill}} \approx \frac{Q}{\lambda - \mu}.$$

बड़ी कतार समय दिलाती है; वह थ्रूपुट पैदा नहीं करती। बैचिंग ट्रांज़ैक्शन की लागत बाँटती है, लेकिन रखे गए रिकॉर्ड को फिर भी मेमोरी और अंततः सेवा चाहिए। ओवरलोड में Sibuna नए घटना रिकॉर्ड छोड़ देता है और उन्हें गिनता है। वह मेमोरी में रखी अवस्था से अनुरोधों के निर्णय लेता रहता है। ऑपरेटर को HTTP सफलता दर के साथ-साथ खोए हुए साक्ष्य की भी निगरानी करनी चाहिए।

अभ्यास 1.2. एक कतार 512 रिकॉर्ड रखती है। आगमन दर $1{,}000/\text{s}$ है और निकासी दर $600/\text{s}$ है। शुरू में खाली कतार अतिरिक्त भार को कितनी देर तक सोख सकती है? उत्तर केवल एक सन्निकटन क्यों है?
संकेत: दरों का अंतर उपयोग करें, फिर बर्स्ट और बैच कमिट पर विचार करें।

वे अपरिवर्तनीय नियम जिन्हें हम आगे ले जाएँगे

1. अविश्वसनीय अनुरोध सर्वर पर असीमित अवस्था नहीं बना सकते।
2. वैध प्रवेश टोकन एप्लिकेशन की अस्वीकृति को बायपास नहीं करता।
3. किसी बफ़र से उधार ली गई हर स्लाइस के रहते वह बफ़र जीवित रहता है।
4. टिकाऊ रीट्राई किसी घटना या उसके प्रतिष्ठा प्रभाव को कई गुना नहीं कर सकता।
5. प्रकाशित नीति स्नैपशॉट तब तक अपरिवर्तनीय रहता है जब तक उसका अंतिम पाठक उसे छोड़ नहीं देता।

शेष पुस्तक वे तंत्र निकालती है जो इन कथनों को सत्य बनाते हैं, और वे परीक्षण भी जो किसी उल्लंघन को उजागर करेंगे। कोई तेज़ पाथ केवल तभी तक उपयोगी है जब तक ये कथन सत्य रहते हैं।

अपरिवर्तनीय नियम समझाएँ. गेट असमिका $c_g < (1 - a)c_o$ को शब्दों में कहें, फिर समझाएँ कि केवल मापी गई सत्यापन लेटेंसी से ऑपरेटर यह क्यों नहीं जान सकता कि गेट चलाना सार्थक है या नहीं।

भाग II

पूर्व कार्य और डिज़ाइन का क्षेत्र

हम *Sibuna* को उन प्रणालियों के बीच रखते हैं जिनसे उसने सीखा: क्रिप्टोग्राफ़ी साहित्य से क्लाइंट पहेलियाँ और अनुक्रमिक कार्य, सिग्नेचर और सिमैंटिक एप्लिकेशन फ़ायरवॉल, *proof-of-work* इंटरस्टिशियल, और होस्टेड *edge* नेटवर्क। फिर हम उन चार उपकरणों की, जिन्हें कोई ऑपरेटर सबसे अधिक एक-दूसरे के विरुद्ध तौलेगा, सुविधा-दर-सुविधा तुलना करते हैं, और दर्ज करते हैं कि कौन-से तथ्य जाँचे गए।

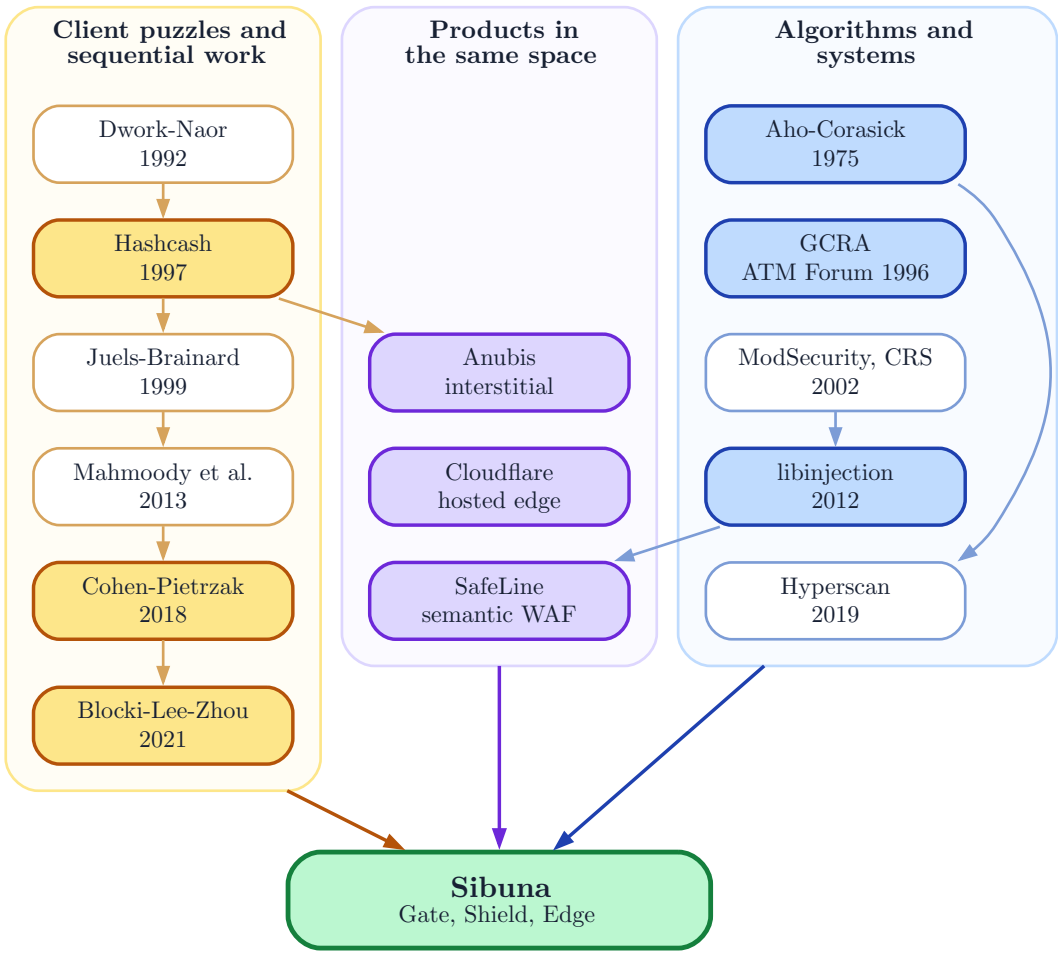
Sibuna कहाँ से आता है

उद्देश्य इस अध्याय के अंत तक हमें यह कर पाना चाहिए: 1992 से 2018 की अनुक्रमिक कार्य वाले निर्माणों तक क्लाइंट पहेलियों की वंश-परंपरा ट्रेस करना, Sibuna द्वारा जोड़े गए तीन आर्किटेक्चरल विचारों के नाम बताना, Sibuna, Anubis, SafeLine और Cloudflare WAF को एक डिज़ाइन क्षेत्र में रखना, और बताना कि तुलना के कौन-से दावे सत्यापित किए गए और कौन-से केवल दस्तावेज़ों से पढ़े गए।

क्लाइंट पहेलियाँ और अनुक्रमिक कार्य

Dwork और Naor के "Pricing via Processing" (CRYPTO 1992) ने प्रस्ताव रखा कि कोई सेवा किसी अनुरोधकर्ता के लिए कार्य करने से पहले एक मध्यम रूप से कठिन, आसानी से जाँची जा सकने वाली गणना माँगे। Back के Hashcash (1997, 2002 में लिखित रूप में) ने पहेली को एक आंशिक हैश व्युत्क्रमण बनाया जिसकी कठिनाई एक पूर्णांक है, और Juels तथा Brainard की क्लाइंट पहेलियों (NDSS 1999) ने इस विचार को कनेक्शन बाढ़ पर लागू किया, जहाँ सर्वर को किसी अनसुलझी पहेली के बारे में कुछ भी याद नहीं रखना चाहिए। तीनों में वही आकार है जो Sibuna बनाए रखता है: सर्वर द्वारा चुना गया स्टेटमेंट, ऐसा समाधान जिसकी लागत समायोज्य है, और ऐसा सत्यापन जो हल करने से कई गुना सस्ता है।

हैश खोज पूरी तरह समानांतर (embarrassingly parallel) है, इसलिए अधिक कोर वाला अनुरोधकर्ता जल्दी पूरा करता है। Mahmoody, Moran और Vadhan (CRYPTO 2013) ने अनुक्रमिक कार्य के प्रमाण परिभाषित किए, जिनमें प्रूवर को कोर की संख्या चाहे जितनी हो, आश्रित कदमों की एक लंबी शृंखला पूरी करनी होती है, और Cohen तथा Pietrzak (EUROCRYPT 2018) ने वह सरल hash-graph निर्माण दिया जिसके सत्यापनकर्ता को केवल $O(t \log N)$ कार्य चाहिए। Blocki, Lee और Zhou (ITC 2021) ने quantum random-oracle model में इसी निर्माण को क्वांटम प्रूवर के विरुद्ध सुदृढ़ सिद्ध किया। भाग III इसी कार्य-परंपरा से Sibuna का टियर दो निकालता है; SID 0006 धारणाएँ दर्ज करता है।



चित्र 2 Sibuna द्वारा जोड़े गए विचारों की वंश-परंपरा। बायाँ स्तंभ क्लाइंट पहेलियों और अनुक्रमिक कार्य की क्रिप्टोग्राफ़ी है, दायाँ स्तंभ वे एल्गोरिदम और प्रणालियाँ हैं जिन पर Sibuna बना है, और बीच का स्तंभ उसी क्षेत्र के उत्पाद हैं; समय ऊपर से नीचे की ओर चलता है। पतले तीर सीधी वंश-परंपरा दिखाते हैं, और भरे हुए बॉक्स वे विचार हैं जिन्हें Sibuna सीधे अपनाता है। स्टोरेज का एक पुराना डिज़ाइन Multi-Paxos (Lamport 1998; SID 0005) पर आधारित था; अब Sibuna अपनी अवस्था SQLite या PostgreSQL में संग्रहीत करता है (SID 0011)।

एप्लिकेशन फ़ायरवॉल

वेब एप्लिकेशन फ़ायरवॉल की पहली पीढ़ी अनुरोध के बाइट का रेगुलर एक्सप्रेशन से मिलान करती थी। ModSecurity (2002) और OWASP Core Rule Set ने इस तरीके को खुला और व्यापक रूप से प्रचलित बनाया, और इसकी लागतें भी परिचित बना दीं: प्रति अनुरोध सैकड़ों बैकट्रैकिंग पैटर्न, हाथ से समायोजित anomaly स्कोर, और सामान्य इनपुट पर false positive की लंबी टेल। libinjection (Hanson, 2012) ने पैटर्न सूचियों की जगह एक टोकनाइज़र रखा जो पूछता है कि कोई स्ट्रिंग SQL के रूप में पार्स होती है या नहीं, जिससे लागत और false positive दोनों घटे। Chaitin का SafeLine इस सिमैंटिक विचार को एक पूरे उत्पाद तक ले जाता है, और Hyperscan (Wang et al., NSDI 2019) ने दिखाया कि लंबे इनपुट पर vectorised लिटरल मिलान को कितना आगे ले जाया जा सकता है। भाग VI समझाता है कि Sibuna ने अपने द्वारा निरीक्षित छोटे फ़ील्ड के लिए रेगुलर एक्सप्रेशन या SIMD इंजन के बजाय सघन ऑटोमेटन और छोटे संरचनात्मक टोकनाइज़र क्यों चुने।

इंटरस्टिशियल और edge

वेबसाइट के सामने proof-of-work इंटरस्टिशियल 2025 में आम हो गया, जब language model के प्रशिक्षण के लिए डेटा जुटाने वाले scraper ट्रैफ़िक ने सामान्य ऑरिजिन चलाना महँगा कर दिया। Anubis (Techaro) सबसे प्रसिद्ध खुला इम्प्लीमेंटेशन है: एक Go रिवर्स प्रॉक्सी जो JavaScript पहेली देता है, सत्र टोकन पर हस्ताक्षर करता है, और बाकी ट्रैफ़िक को आगे जाने देता है। Sibuna की Gate सतह यही समस्या एक स्टेटलेस चैलेंज, एक सममित टोकन और कार्य के दूसरे टियर से हल करती है; नीचे की तुलना दर्ज करती है कि दोनों कहाँ भिन्न हैं।

पैमाने के दूसरे छोर पर, Cloudflare जैसे होस्टेड edge निरीक्षण, रेट लिमिटिंग और बॉट स्कोरिंग को सैकड़ों points of presence में एक वैश्विक control plane के साथ रखते हैं। Sibuna का Edge डिप्लॉयमेंट यह आकार उधार लेता है (एक साझा नीति और प्रतिष्ठा plane जो स्थानीय, अपरिवर्तनीय स्नैपशॉट को भरता है) और फिर भी self-hosted रहता है, इसीलिए स्टोरेज परत किसी सेवा कॉल के बजाय ऑपरेटर द्वारा चलाया गया डेटाबेस है (एक नोड पर एक SQLite फ़ाइल, क्लस्टर द्वारा साझा एक PostgreSQL डेटाबेस), जिसे अनुरोध पाथ से बाहर एक स्टोरेज थ्रेड पढ़ता है।

एक बाइनरी में तीन विचार

Sibuna प्रवेश के लिए ब्राउज़र कार्य चैलेंज, सीमित एप्लिकेशन-पेलोड निरीक्षण, और स्थानीय नीति स्नैपशॉट प्रकाशित करने वाले साझा control plane को जोड़ता है। Gate और Shield उत्पाद की दो सतहें हैं; स्टोरेज और वितरित डिप्लॉयमेंट दोनों के लिए विकल्प हैं।

गुण	Gate	Shield
प्रवेश	नेटिव Hashcash या अनुक्रमिक कार्य का सत्यापन	वही
सत्र	कुंजी-युक्त BLAKE3; वैकल्पिक Ed25519	वही
स्थानीय नियंत्रण	नियम, CIDR, GCRA, प्रतिबंध	वही
निरीक्षण	अक्षम	सीमित सिग्नेचर और संरचनात्मक टोकनाइज़र
अवस्था	वैकल्पिक स्थायी नीति और प्रतिष्ठा, क्लस्टर में साझा	निरीक्षण की फ़ॉरेंसिक जानकारी भी

ब्राउज़र सॉल्वर उन्हीं प्रमाण स्रोतों से कंपाइल होता है जिनसे नेटिव सत्यापनकर्ता। सघन लिटरल मिलान बैकट्रैकिंग से बचता है और स्कैन का कार्य इनपुट की लंबाई में रैखिक रखता है। अपरिवर्तनीय नीति स्नैपशॉट अनुरोध हैंडलिंग को SQL से और डेटाबेस की उपलब्धता से अलग रखते हैं। इनमें से कोई भी चुनाव सार्वभौमिक एल्गोरिदमिक इष्टतमता स्थापित नहीं करता: छोटे हेडर, लंबी बॉडी, पैटर्न की विविधता, कोर की संख्या और cache का आकार इन समझौतों को बदल देते हैं।

अपनाई गई डिज़ाइन शर्तें

1. अनुरोध पाथ पर कोई हीप आवंटन नहीं; हर टेबल की एक निश्चित क्षमता और भर जाने पर एक घोषित व्यवहार है।
2. चैलेंज जारी करने में कुछ भी नहीं लिखा जाता; केवल चुकाया गया समाधान मेमोरी घेरता है।
3. सत्र प्रवेश देता है; वह किसी अनुरोध को कभी निरीक्षण या स्पष्ट अस्वीकृति से छूट नहीं देता।
4. हर दावा की गई संख्या या तो किसी कमिट किए गए हार्नेस से मापी गई है या मॉडल के रूप में चिह्नित है।
5. क्रिप्टोग्राफ़िक चुनाव किसी प्रकाशन-स्थल और प्रमाण का हवाला देते हैं, और क्वांटम के प्रति अपनी स्थिति बताते हैं।

सुविधाओं की तुलना

नीचे की चार तालिकाएँ Sibuna की तुलना उन तीन उपकरणों से करती हैं जिनका उल्लेख अक्सर उसके साथ होता है। अन्य उत्पादों के बारे में तथ्य सितंबर 2026 में उनके सार्वजनिक दस्तावेज़ों और रिलीज़ सामग्री से पढ़े गए: Anubis 1.27.0 (MIT लाइसेंस, Go), SafeLine community edition 9.x (बंद डिटेक्टर इमेज के चारों ओर GPL-3.0 प्रबंधन कोड, Docker Compose), और Cloudflare WAF का डेवलपर दस्तावेज़ीकरण। Anubis को बेंचमार्क होस्ट पर चलाया भी गया; SafeLine और Cloudflare को नहीं (भाग VIII कारण बताता है)। डैश का अर्थ है कि उत्पाद यह सुविधा नहीं देता; प्लान के नाम का अर्थ है कि सुविधा मौजूद है लेकिन केवल उस प्लान पर। Sibuna का हर मापन उस कमिट की गई परिणाम फ़ाइल का नाम देता है जिससे वह आया है। TLS फ़िंगरप्रिंट वाली पंक्ति अक्टूबर 2026 में Anubis के changelog (1.21.0 और 1.26.0), SafeLine के release notes (8.10.0 और 9.2.0) और Cloudflare के पेज "JA3/JA4 fingerprint" से मिलाकर जाँची गई।

तालिका 3 सुविधाओं की तुलना, भाग 1: डिप्लॉयमेंट

सुविधा	Sibuna	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
भाषा, रनटाइम	Rust, कोई garbage collector नहीं; स्टोरेज बंद होने पर कोई बाहरी क्रेट 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
होस्ट पर संसाधन	35.7 MiB बाइनरी (`benchmarks/results/comparison-latest.json`); 5.9 MiB निष्क्रिय RSS (`benchmarks/results/primitives-latest.json`); लोड में अधिकतम 16.0 MiB पीक RSS (`benchmarks/results/product-latest.md`)	38.1 MiB बाइनरी; 21 MiB निष्क्रिय RSS; Linux fixture, कंसोल बंद	न्यूनतम 1 CPU कोर, 1 GB RAM, 5 GB डिस्क, Docker 20.10+	अपने परिसर में कुछ नहीं

तालिका 4 सुविधाओं की तुलना, भाग 2: प्रवेश और सत्र

सुविधा	Sibuna	Anubis 1.27	SafeLine CE 9	Cloudflare WAF
Proof-of-work प्रवेश	Hashcash (बिट-स्तर) और Cohen-Pietrzak अनुक्रमिक कार्य, प्रति नियम चुना गया; JavaScript फ़ॉलबैक के साथ WebAssembly	हेक्स nibble में SHA-256 Hashcash (`fast`, `slow`); कार्य-रहित `metarefresh` और `preact` चैलेंज	JavaScript anti-bot चैलेंज और CAPTCHA; कार्य से बँधा नहीं	Managed challenge, JS challenge, Turnstile; Bot Fight Mode एक गणना चैलेंज जारी करता है
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 टैग, चुकाए गए कार्य से बँधी 75 अक्षरों की कुकी; Ed25519 वैकल्पिक	Ed25519 से हस्ताक्षरित JSON Web Token; HS512 वैकल्पिक	स्टैक द्वारा जारी कुकी	`cf_clearance` कुकी
डिफ़ॉल्ट टोकन की पोस्ट-क्वांटम स्थिति	सममित: केवल Grover की द्विघाती गति-वृद्धि लागू होती है	Ed25519: Shor के एल्गोरिदम से टूट जाता है	दस्तावेज़ों में नहीं	दस्तावेज़ों में नहीं
नीति भाषा	JSON नियम: पाथ, User-Agent, हेडर, IPv4/IPv6 CIDR, JA4 फ़िंगरप्रिंट; ALLOW, DENY, CHALLENGE, थ्रेशोल्ड के साथ WEIGH; डेटाबेस से डायनामिक नियम	YAML/JSON bots: UA, पाथ, हेडर पर regex; CIDR; CEL एक्सप्रेशन; थ्रेशोल्ड के साथ WEIGH	कंसोल में परिभाषित कस्टम नियम, ACL, IP समूह	Rules language एक्सप्रेशन; प्लान के अनुसार *5 / 20 / 100 / 1000* कस्टम नियम
TLS फ़िंगरप्रिंट (JA4)	नियम JA4 फ़िंगरप्रिंट का सटीक या प्रीफ़िक्स से मिलान करते हैं; कोई विश्वसनीय इनग्रेस उसकी गणना करके उसे एक हेडर (`--ja4-header`) में फ़ॉरवर्ड करता है; Sibuna स्वयं उसकी गणना नहीं करता	दस्तावेज़ों में नहीं; नीतियाँ HTTP फ़िंगरप्रिंट JA4H का संदर्भ दे सकती हैं, जिसकी गणना करके Anubis उसे `X-Http-Fingerprint-JA4H` हेडर के रूप में जोड़ता है	अपने ही TLS टर्मिनेशन पर गणना; आक्रमण के विवरण में दिखता है; कस्टम नियम उसका मिलान करते हैं (equals, not equals); एक बिल्ट-इन नियम उसकी दुर्भावनापूर्ण JA4 इंटेलिजेंस में	WAF कस्टम नियमों और analytics में JA3 और JA4 फ़िंगरप्रिंट *(Bot Management* के साथ *Enterprise)*

सुविधा	Sibuna	Anubis 1.27	SafeLine CE 9	Cloudflare WAF
			दर्ज फ़िंगरप्रिंट अस्वीकार करता है	

तालिका 5 सुविधाओं की तुलना, भाग 3: निरीक्षण और अवस्था

सुविधा	Sibuna	Anubis 1.27	SafeLine CE 9	Cloudflare WAF
SQLi, XSS, RCE, traversal निरीक्षण	Shield: टैग किया गया ऑटोमेटन और संरचनात्मक टोकनाइज़र, बॉडी के पहले 8 KB	-	OWASP श्रेणियों पर सिमैंटिक विश्लेषण इंजन, साथ में SSRF, XXE, CRLF	Managed rulesets *(Pro* और ऊपर*)*; OWASP Core Ruleset; attack score *(Business* और ऊपर*)*
रेट लिमिटिंग	प्रति क्लाइंट GCRA, एक नोड पर स्थानीय	- (सिस्टम लोड CEL नियमों को उपलब्ध है)	प्रति IP, पाथ, सत्र	प्लान के अनुसार *1 / 2 / 5 / 100* नियम; Enterprise से नीचे केवल IP कुंजियाँ
प्रतिबंध और प्रतिष्ठा	हनीपॉट प्रतिबंध; क्लस्टर में साझा प्रतिष्ठा टेबल से लोड किया गया प्रतिष्ठा trie	DNSBL; Thoth के माध्यम से ASN और GeoIP *(*सशुल्क, बंद*)*	IP समूह और ब्लैकलिस्ट; threat intelligence *(Pro)*	IP सूचियाँ; managed lists और bot score *(Enterprise)*
फ़ॉरेंसिक	SQLite या PostgreSQL: घटनाओं पर full-text खोज, vector कैंपेन क्लस्टरिंग	केवल Prometheus मेट्रिक	कंसोल के साथ PostgreSQL आक्रमण लॉग	Security Events; Free पर सैंपल किए गए
मल्टी-नोड	नीति और प्रतिष्ठा के लिए हर नोड द्वारा साझा एक PostgreSQL डेटाबेस; साझा सीड; sticky चैलेंज सत्यापन	साझा signing कुंजी; साझा Valkey स्टोर	प्रति होस्ट एक स्टैक	वैश्विक anycast

तालिका 6 सुविधाओं की तुलना, भाग 4: संचालन

सुविधा	Sibuna	Anubis 1.27	SafeLine CE 9	Cloudflare WAF
मेट्रिक	`/__sibuna/metrics` पर Prometheus	अलग पोर्ट पर Prometheus	कंसोल डैशबोर्ड	डैशबोर्ड और analytics
TLS टर्मिनेशन	- (इनग्रेस)	- (इनग्रेस)	**हाँ**	**हाँ**

सुविधा	Sibuna	Anubis 1.27	SafeLine CE 9	Cloudflare WAF
प्रबंधन कंसोल	वैकल्पिक (opt-in); SID 0007 Committed	-	**हाँ**	**हाँ**
बेंचमार्क होस्ट पर मापा गया	**हाँ** (भाग VIII)	**हाँ**	- (केवल Docker)	- (होस्टेड)

चयन के निर्णय में सबसे अधिक भार तीन अंतरों का है।

- कार्य कहाँ जाता है। Anubis और Sibuna Gate ऑरिजिन के कुछ भी करने से पहले क्लाइंट से कीमत चुकवाते हैं। SafeLine और Cloudflare पहले निरीक्षण करते हैं और चुनिंदा रूप से चैलेंज करते हैं; उनका डिफ़ॉल्ट रुख प्रवेश देकर छानने का है। Sibuna Shield दोनों करता है, इसी क्रम में: निरीक्षण उस अनुरोध को भी अस्वीकार कर सकता है जिसके पास पहले से वैध सत्र है।
- सर्वर क्या याद रखता है। Anubis चैलेंज की अवस्था एक स्टोर (memory, bbolt, Valkey या S3) में रखता है और Ed25519 से JSON Web Token पर हस्ताक्षर करता है; Sibuna एक स्वयं-प्रमाणित रिकॉर्ड जारी करता है और केवल हल किए गए टैग एक निश्चित टेबल में याद रखता है। SafeLine आक्रमण लॉग PostgreSQL में रखता है; Cloudflare सब कुछ रखता है, Free प्लान पर सैंपल की गई दृश्यता के साथ।
- ऑपरेटर को क्या चलाना पड़ता है। Sibuna वैकल्पिक SQLite फ़ाइल, या क्लस्टर द्वारा साझा PostgreSQL डेटाबेस, के साथ एक बाइनरी है; Anubis एक Go बाइनरी है; SafeLine कम से कम एक कोर, एक गीगाबाइट मेमोरी और पाँच गीगाबाइट डिस्क वाले Linux होस्ट पर Docker डेमन के पीछे सात कंटेनर है; Cloudflare एक DNS बदलाव और एक सब्सक्रिप्शन है।

उत्पाद की वर्तमान सीमा Sibuna इनग्रेस पर TLS टर्मिनेशन डिप्लॉयमेंट प्रॉक्सी पर छोड़ता है, और न तो प्रशिक्षित मॉडल से बॉट का स्कोर निकालता है और न ही सशुल्क सिग्नेचर प्रकाशित करता है। वह उस JA4 फ़िंगरप्रिंट का मिलान करता है जिसकी गणना कोई विश्वसनीय इनग्रेस करता है; वह स्वयं उसकी गणना नहीं कर सकता (SID 0014, भाग IX)। उसका वैकल्पिक कंसोल प्रमाणित डैशबोर्ड, देश की जानकारी, नीति संपादन, जाँच और क्लस्टर प्रबंधन देता है। देश का लुकअप और स्टोरेज का कार्य अनुरोध वर्गीकरण से बाहर चलते हैं। SID 0007 Committed है, और कंसोल अपने ब्राउज़र वर्कफ़्लो पास करता है (SID 0010)। लोड में आठ सक्रिय डैशबोर्ड के साथ उसका औपचारिक प्रदर्शन-पृथक्करण गेट अभी मापा नहीं गया है; भाग VIII दर्ज करता है कि क्या मापा गया है।

अभ्यास 2.1. समझाएँ कि एक ही सत्यापनकर्ता स्रोत को नेटिव कोड और WebAssembly में कंपाइल करने से प्रोटोकॉल का विचलन क्यों घटता है। किन गुणों के लिए अब भी स्वतंत्र परीक्षण या क्रिप्टोग्राफ़िक समीक्षा चाहिए?

अभ्यास 2.2. केवल तुलना तालिकाओं का उपयोग करके वह सबसे छोटा डिप्लॉयमेंट लिखें जो किसी स्टैटिक साइट को (a) proof-of-work गेट, (b) SQL-injection निरीक्षण, और (c) क्लस्टर-व्यापी प्रतिबंध दे। किन पंक्तियों ने हर चुनाव को बाध्य किया?
संकेत: "किस रूप में चलता है" और "मल्टी-नोड" पंक्तियों से शुरू करें।

अपरिवर्तनीय नियम समझाएँ. वर्णन करें कि Gate, Shield और वैकल्पिक साझा स्टोरेज प्रवेश, निरीक्षण और नीति वितरण को कैसे बाँटते हैं। फिर तुलना तालिकाओं की एक ऐसी पंक्ति बताएँ जिसे हम किसी ग्राहक को बताने से पहले दोबारा सत्यापित करना चाहेंगे, और बताएँ कि हम उसे कैसे सत्यापित करेंगे।

भाग III

क्रिप्टोग्राफ़ी अनुसंधान से आधार

हम *Sibuna* के दो *proof-of-work* टियर के पीछे के लागत मॉडल निकालते हैं, *Cohen-Pietrzak* के अनुक्रमिक कार्य के प्रमाण *(Proof of Sequential Work)* को उसके सुरक्षा तर्क के साथ प्रस्तुत करते हैं, और चैलेंज तथा सत्र के लिए कुंजी-युक्त हैश प्रमाणीकरण का औचित्य बताते हैं। *SID 0006* धारणाएँ और संदर्भ दर्ज करता है।

Hashcash: टियर एक का गणित

उद्देश्य इस अध्याय के अंत तक हमें यह कर पाना चाहिए: बिट-स्तर की Hashcash पहेली का अपेक्षित कार्य, प्रसरण और टेल प्रायिकता बताना, समझाना कि सर्वर की सत्यापन लागत हमलावर के लाभ को कैसे सीमित करती है, और उन दो सीमाओं का वर्णन करना जो दूसरे टियर की प्रेरणा देती हैं।

परिभाषा और लागत

चैलेंज स्ट्रिंग C और b बिट की कठिनाई के लिए Sibuna की टियर एक पहेली यह है: ऐसा $N \in \mathbb{N}$ खोजें कि SHA-256 $(C \,\|\, ":" \,\|\, \mathrm{dec}\,(N))$ में b शुरुआती शून्य बिट हों। नॉनस को दशमलव रूप में लिखना जानबूझकर है; इससे WebAssembly सॉल्वर, JavaScript फ़ॉलबैक और नेटिव सत्यापनकर्ता के बीच वायर प्रारूप बिना किसी बाइनरी एन्कोडिंग चरण के एक समान रहता है।

कठिनाई की इकाई एक अपेक्षित हैश प्रयास (hash trial) है, compression फ़ंक्शन की कॉल नहीं। एक इनपुट कई SHA-256 ब्लॉक घेर सकता है। प्रीफ़िक्स की पूर्व-गणना भी प्रति नॉनस कार्य बदल सकती है। वायर नियम डाइजेस्ट प्रेडिकेट ही रहता है; इम्प्लीमेंटेशन को स्टेटमेंट और नॉनस एन्कोडिंग पर सहमत होना चाहिए, अनुमानित CPU लागत पर नहीं।

स्वतंत्र आदर्श डाइजेस्ट के लिए प्रयास संख्या X, $p = 2^{-b}$ के साथ ज्यामितीय है। इसकी माध्यिका $\lceil \ln(1/2)/\ln(1-p) \rceil$ है, जो छोटे p के लिए लगभग $2^b \ln 2$ है, और इसका प्रसरण $(1-p)/p^2$ है। सत्यापनकर्ता खोज करने के बजाय एक उम्मीदवार डाइजेस्ट की जाँच करता है। पार्सिंग, चैलेंज प्रमाणीकरण, फ़िंगरप्रिंट जाँच, spent-state में प्रविष्टि और रिस्पॉन्स अतिरिक्त कार्य हैं। किसी प्रिमिटिव के समय-माप को उलटकर सर्वर की बाढ़ सहने की क्षमता नहीं निकाली जा सकती।

खोज की दो सीमाएँ

पहली सीमा प्रसरण है: लंबी खोज का अर्थ यह नहीं कि वर्कर खराब है। दूसरी सीमा समानांतरता है: स्वतंत्र नॉनस प्रयास एक साथ चल सकते हैं। अधिक कोर खोज में बीता समय घटाते हैं, भले ही अपेक्षित कुल प्रयास संख्या वही रहे। इसलिए प्रवेश नीति के लिए हार्डवेयर थ्रूपुट मायने रखता है। न कोई दावा की गई हैश दर और न कोई अकेला बेंचमार्क हर अनुरोधकर्ता के लाभ को सीमित कर सकता है।

अभ्यास 3.0. ज्यामितीय टेल का उपयोग करके 4.6×2^b से अधिक प्रयासों की आवश्यकता की प्रायिकता का अनुमान लगाएँ। नॉनस स्पेस को कई वर्करों में बाँटने से अपेक्षित प्रयास संख्या क्यों नहीं बदलती?
संकेत: कुल प्रयास गिनें, बीते हुए सेकंड नहीं।

अनुक्रमिक कार्य का प्रमाण: टियर दो

उद्देश्य Cohen-Pietrzak लेबलिंग को समझें: यह अनुक्रमिक गणना को क्यों अनिवार्य करती है, सत्यापनकर्ता क्या जाँचता है, सशर्त सैंपलिंग सीमा $(1-\alpha)^t$ और उसकी सीमाएँ, पोस्ट-क्वांटम परिणाम, प्रूवर की मेमोरी सीमा, और वर्क-बिट मैपिंग के पीछे का कैलिब्रेशन डेटा।

अनुक्रमिक कार्य क्यों

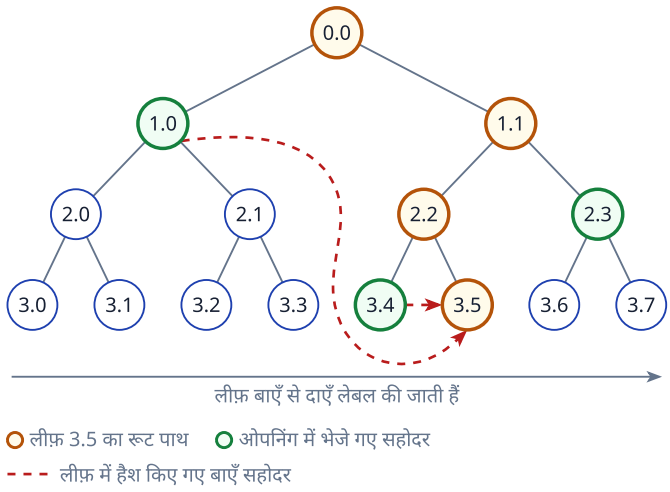
अनुक्रमिक कार्य वाला निर्माण गणनाओं के बीच निर्भरताएँ लाता है: बाद के लेबल को पहले बना मान चाहिए। यह कुल कार्य से अलग एक मात्रा को संबोधित करता है, यानी गणना की गहराई। स्वतंत्र पहेलियाँ फिर भी समानांतर चल सकती हैं, और हार्डवेयर की हैश लेटेंसी फिर भी मायने रखती है।

Sibuna, Cohen और Pietrzak द्वारा वर्णित हैश-आधारित निर्माण का अनुसरण करता है। इसके ग्राफ़ में गहराई n पर $2^{n+1}-1$ लेबल होते हैं, लेकिन लेबल की निश्चित संख्या ब्राउज़र की निश्चित लेटेंसी नहीं है: लेबल कई compression ब्लॉक में फैल सकते हैं, ओपनिंग सबट्री की दोबारा गणना कर सकती हैं, और शेड्यूलिंग भिन्नता जोड़ती है। oracle-model विश्लेषणों से मिले सुरक्षा कथनों को हर ठोस इम्प्लीमेंटेशन या विरोधी के बारे में बिना शर्त दावे में नहीं बदलना चाहिए।

निर्माण

मान लें n गहराई है, और उस पूर्ण बाइनरी ट्री पर विचार करें जिसके नोड अधिकतम n लंबाई की बाइनरी स्ट्रिंग से नामित हैं (रूट ε, चाइल्ड $v0$ और $v1$, लीफ़ की लंबाई n)। हर नोड v एक 32-बाइट लेबल $\ell_v = H(\chi, v, \text{parents }(v))$ रखता है, जहाँ χ चैलेंज स्टेटमेंट है और H वह SHA-256 है जिसमें χ एक बार सोखा गया है (Hashcash सॉल्वर वाली ही प्रीफ़िक्स तरकीब):

- आंतरिक नोड अपने दो चाइल्ड का हैश करता है: $\ell_v = H(\chi, v, \ell_{v0}, \ell_{v1})$;
- लीफ़ $u = u_1 \ldots u_n$ उन सभी स्थानों पर अपने पूर्वजों के बाएँ सहोदरों का हैश करती है जहाँ रूट से आने वाला पाथ दाईं ओर मुड़ा: $\ell_u = H(\chi, u, \{\ell_{u_1 \ldots u_{i-1} 0} : u_i = 1\})$।



चित्र 3 गहराई-3 का ट्री: लीफ़ 3.5 की ओपनिंग लीफ़ लेबल और तीन सहोदर लेबल भेजती है; डैश वाले किनारे बाएँ सहोदर की वे निर्भरताएँ हैं जो लेबलिंग को अनुक्रमिक बनाती हैं।

बाएँ-सहोदर वाले किनारे ही गणना को अनुक्रमिक बनाती हैं: लीफ़ u का लेबल तब तक नहीं बन सकता जब तक उसके बाईं ओर का हर सबट्री पूरा न हो जाए, इसलिए depth-first post-order traversal अनिवार्य हो जाता है और रूट लेबल $\varphi = \ell_\varepsilon$ क्रम से सभी $2^{n+1} - 1$ लेबल पर निर्भर करता है।

ओपनिंग। φ की गणना के बाद प्रूवर Fiat-Shamir से t लीफ़ निकालता है, $\gamma_i = H(\chi \,\|\, \varphi \,\|\, i) \bmod 2^n$, और हर एक के लिए लीफ़ लेबल तथा रूट पाथ पर पड़ने वाले n सहोदर लेबल भेजता है। सत्यापनकर्ता उन सहोदरों से लीफ़ की दोबारा गणना करता है जो उसके पैरेंट हैं, पाथ से ऊपर की ओर हैश करता है, और जाँचता है कि वह φ तक पहुँचता है। सत्यापनकर्ता की लागत सीमित वर्कस्पेस के साथ $O(tn)$ लेबल गणनाएँ है। एक लेबल गणना कई हैश ब्लॉक में फैल सकती है; लेबल गिनना compression कॉल गिनना नहीं है।

> **इम्प्लीमेंटेशन: `posw::verify_opening`** खोली गई एक लीफ़ की उसके सहोदर लेबल से दोबारा गणना करता है और रूट तक पाथ पर चलता है। `crates/pow/src/posw.rs` में (क्रेट `sibuna-pow`)।

```
fn verify_opening(base: &Sha256, n: u8, phi: &Label, gamma: u32, opening:
&[Label]) -> bool {
    let Some((leaf, siblings)) = opening.split_first() else {
        return false;
    };
    // `siblings` गहराई n से गहराई 1 तक चलता है; उलटने पर यह गहराइयों 1..=n से जोड़ा बनाता है।
    let mut hasher = node_hasher(base, n, gamma);
    for (d, sibling) in (1..=n).zip(siblings.iter().rev()) {
        if turns_right(gamma, n, d) {
            hasher.update(sibling);
        }
    }
    if hasher.finalize() != *leaf {
        return false;
    }
    let mut current = *leaf;
    for (d, sibling) in (1..=n).rev().zip(siblings) {
        let mut parent = node_hasher(base, d - 1, gamma >> (n - d + 1));
        if turns_right(gamma, n, d) {
            parent.update(sibling);
            parent.update(&current);
        } else {
            parent.update(&current);
            parent.update(sibling);
        }
        current = parent.finalize();
    }
    current == *phi
}
```

सैंपलिंग, सुदृढ़ता और उनकी सीमाएँ

एक निश्चित कमिटमेंट पर विचार करें जिसमें स्वतंत्र रूप से पहचाना जा सकने वाला खराब अंश α है। यदि हर ओपनिंग समान रूप से और स्वतंत्र रूप से सैंपल करती है, तो t ओपनिंग में हर खराब स्थान के छूट जाने की संभावना $(1 - \alpha)^t$ है। यह प्रारंभिक गणना समझाती है कि अधिक ओपनिंग क्यों मदद करती हैं। यह अपने आप में अनुक्रमिक कार्य वाले प्रोटोकॉल का सुरक्षा प्रमाण नहीं है।

हल किया गया उदाहरण: एक सशर्त सैंपलिंग सीमा निश्चित खराब अंश $\alpha = 0.1$ और $t = 16$ स्वतंत्र ओपनिंग के साथ हर खराब स्थान के छूट जाने की संभावना $0.9^{16} \approx 0.185$ है। 32 ओपनिंग के साथ यह लगभग 0.034 है। आधार-धारणा सैंपल चुने जाने से पहले कमिटमेंट को निश्चित कर देती है। यदि कोई प्रूवर कई रूट आज़मा (grind) सकता है, चुन सकता है कि कौन-सा कमिटमेंट प्रकट करे, या विफलताओं को परस्पर संबद्ध कर सकता है, तो यह गणना उस रणनीति का हिसाब नहीं रखती।

संदर्भित अनुक्रमिक कार्य वाले निर्माण का oracle model में सुरक्षा विश्लेषण है। Sibuna के domain separation, पैरामीटर मैपिंग, truncation और इम्प्लीमेंटेशन को फिर भी निर्माण की धारणाओं के विरुद्ध जाँचना होगा। Fiat-Shamir सैंपलिंग ओपनिंग को कमिटमेंट से दोबारा बनाने योग्य बनाती है; केवल दोबारा बनाने योग्य होना grinding के प्रति प्रतिरोध सिद्ध नहीं करता। हम कोई बिना शर्त दावा नहीं करते कि धोखा देना कभी सस्ता नहीं होता।

अभ्यास 3.1. यदि कोई हमलावर g स्वतंत्र कमिटमेंट आज़माता है, जिनमें से हर एक q प्रायिकता से स्वीकार होता है, तो कम से कम एक के स्वीकार होने की प्रायिकता निकालें। उस अभिव्यक्ति में कौन-सी लागतें छूटी हुई हैं? संकेत*:* पहले यह प्रायिकता निकालें कि सभी g विफल हों।

प्रूवर की मेमोरी

एक सीधा-सादा प्रूवर हर लेबल ($2^{n+1} - 1$ लेबल) संग्रहीत करता है। Sibuna का प्रूवर पहले पास के दौरान मौजूदा पाथ के बाएँ-सहोदर लेबल का एक स्टैक ($n + 1$ लेबल) रखता है और केवल ऊपर के $m = \min(n, 10)$ स्तर ($2^{m+1} - 1$ लेबल, अधिकतम 64 KB) बनाए रखता है। किसी लीफ़ को खोलने के लिए वह लीफ़ के गहराई m वाले पूर्वज के सबट्री की दोबारा गणना करता है, यानी $2^{n-m+1} - 1$ हैश, और रखे गए स्तरों से स्टैक को प्रारंभ करता है। प्रूवर की कुल लागत $2^{n+1} - 1 + t\,(2^{n-m+1} - 1)$ हैश है, और आउटपुट प्रमाण सहित पूरा वर्कस्पेस 100 KB से कम है: ब्राउज़र टैब और नेटिव परीक्षण एक ही `Workspace` प्रकार (`sibuna_pow::posw::Workspace`) साझा करते हैं।

पैरामीटर समय-माप नहीं हैं

Sibuna वर्क बिट्स को $n = b - 3$ से गहराई में बदलता है, जिसे समर्थित अंतराल $[4, 24]$ में सीमित किया जाता है। यह कार्य के पैमाने की एक परिपाटी है, एल्गोरिदम के बीच बराबर वास्तविक समय की गारंटी नहीं। एक लीफ़ कई पैरेंट लेबल का हैश कर सकती है, जबकि Hashcash बार-बार एक नॉनस परखता है। कंपाइलर, ब्राउज़र, प्रोसेसर और पैरामीटर के चुनाव, सभी उनकी सापेक्ष लागतों को प्रभावित करते हैं। कमिट किया गया प्रिमिटिव बेंचमार्क सत्यापन को मापता है; वह ब्राउज़र में हल की लेटेंसी स्थापित नहीं करता।

प्रमाण का आकार $32(1 + t(n + 1))$ बाइट है। सोलह ओपनिंग के साथ गहराई 13 पर यह $32(1 + 16 \times 14) = 7200$ बाइट है। ओपनिंग की संख्या बढ़ाने से प्रमाण और सत्यापनकर्ता का कार्य, दोनों बढ़ते हैं, भले ही ट्री बनाने की प्रमुख लागत वही रहे।

अभ्यास 3.2. 16 ओपनिंग के साथ गहराई 17 पर प्रमाण का आकार निकालें। फिर ओपनिंग की संख्या दोगुनी करें। कौन-सा पद दोगुना होता है और कौन-सा स्थिर रहता है?

निर्माण का चुनाव

प्रासंगिक तुलना संसाधन बजट की है, किसी एल्गोरिदम की प्रतिष्ठा की नहीं। प्रूवर की मेमोरी, सत्यापनकर्ता का सबसे खराब स्थिति वाला कार्य, प्रमाण के बाइट, और दावा किए गए गुण के लिए आवश्यक धारणाएँ दर्ज करें। memory-hard खोज उपयोगी हो सकती है, लेकिन जिस सत्यापनकर्ता को महँगा कार्य दोहराना पड़े, वह हमलावर को सर्वर के संसाधन खर्च कराने

की एक और जगह देता है। सार्वजनिक रूप से सत्यापित की जा सकने वाली delay निर्माण साझा सीक्रेट वाले गेट से भिन्न विश्वास मॉडल की सेवा करते हैं।

Sibuna सस्ता हैश सत्यापन और हैश-आधारित अनुक्रमिक कार्य का विकल्प बनाए रखता है। अनुक्रमिक कार्य के लिए अधिक प्रमाण बाइट और अधिक सावधान पैरामीटर विश्लेषण चाहिए। कोई भी चुनाव महँगे सत्यापन में प्रवेश से पहले विकृत प्रमाणों को अस्वीकार करने की आवश्यकता को नहीं हटाता।

अपरिवर्तनीय नियम समझाएँ. प्रमाणीकरण ट्री, जो सिद्ध करता है कि कोई लेबल किसी कमिटमेंट का हिस्सा है, और निर्भरता वाले किनारों, जो यह तय करते हैं कि लेबल की गणना कैसे होती है, के बीच का अंतर समझाएँ।

सममित प्रमाणीकरण: चैलेंज और सत्र

उद्देश्य समझाएँ कि जब जारीकर्ता और सत्यापनकर्ता एक ही विश्वास क्षेत्र में हों, तब कुंजी-युक्त pseudorandom फ़ंक्शन ही सही प्रिमिटिव क्यों है; 128-बिट टैग के लिए जालसाज़ी की सीमा बताएँ; स्टेटलेस चैलेंज रिकॉर्ड और spent-set सीमा का वर्णन करें; और key schedule पढ़ें।

जारीकर्ता ही सत्यापनकर्ता है

डिजिटल हस्ताक्षर सार्वजनिक कुंजी रखने वाले किसी को भी सत्यापन करने देता है, जो तब सही उपकरण है जब सत्यापनकर्ताओं को टोकन बनाने (mint) में सक्षम नहीं होना चाहिए। Sibuna का सत्यापनकर्ता ही जारीकर्ता है (या एक सीड साझा करने वाला क्लस्टर)। pseudorandom फ़ंक्शन से बना message authentication code उस विश्वास क्षेत्र के भीतर प्रमाणीकरण देता है। यह सार्वजनिक सत्यापन नहीं देता: MAC कुंजी रखने वाला कोई भी टोकन बना भी सकता है। बेंचमार्क अध्याय लागू किए गए टोकन संचालनों की तुलना करता है।

लेम्मा 3 (जालसाज़ी की सीमा) keyed मोड में BLAKE3 को PRF के रूप में मॉडल करने और 16-बाइट टैग के साथ, q सत्यापन क्वेरी करने वाला विरोधी अधिकतम $q \cdot 2^{-128} + \varepsilon_{\mathrm{PRF}}$ प्रायिकता से वैध टोकन जाली बना पाता है। प्रति सेकंड दस लाख क्वेरी पर यह प्रति सेकंड 2^{-108} है।

दीर्घकाल के लिए एक दूसरा परिणाम मायने रखता है: Ed25519 elliptic-curve समूह में discrete logarithm पर टिका है, जिसे Shor का एल्गोरिदम तोड़ देता है; कुंजी-युक्त हैश केवल हैश फ़ंक्शन पर टिका है, जिसके विरुद्ध क्वांटम एल्गोरिदम अधिकतम द्विघाती गति-वृद्धि देते हैं। Sibuna उन डिप्लॉयमेंट के लिए Ed25519 को एक विकल्प (`--token-scheme ed25519`) के रूप में रखता है जिन्हें टोकन न बना सकने वाले सत्यापनकर्ता चाहिए।

कुंजी अनुसूची (key schedule)

एक 32-बाइट मास्टर सीड (`--secret-file`, `SIBUNA_SECRET` से, या एक नए यादृच्छिक मान से) कुंजी-युक्त BLAKE3 और एक domain स्ट्रिंग के साथ हर उद्देश्य की कुंजी निकालता है: टोकन, चैलेंज, Ed25519 और फ़िंगरप्रिंट कुंजियाँ, और वह आवश्यकता कुंजी जो आवश्यकता टिकट को सील करती है (भाग IV)।

इम्प्लीमेंटेशन: `Keys::derive` सीड से पाँच उद्देश्य-कुंजियाँ निकालता है। `crates/challenge/src/keys.rs` में (क्रेट `sibuna-challenge`)। `Keys` और `Seed` छोड़े जाने (drop) पर अपने बाइट शून्य कर देते हैं, और उनका `Debug` आउटपुट छिपाया (redact) जाता है।

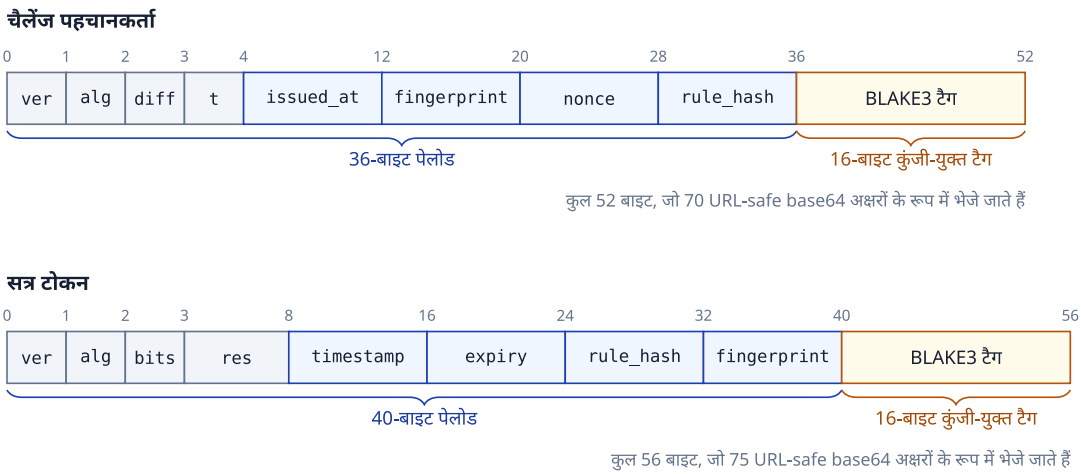
```rust
pub fn derive(seed: &Seed) -> Keys {
    Keys {
        token: subkey(seed, "sibuna/token/v1"),
        challenge: subkey(seed, "sibuna/challenge/v1"),
        ed25519_seed: subkey(seed, "sibuna/ed25519/v1"),
        fingerprint: subkey(seed, "sibuna/fingerprint/v1"),
        requirement: subkey(seed, "sibuna/requirement/v1"),
    }
}

fn subkey(seed: &Seed, purpose: &str) -> [u8; 32] {
    blake3::keyed_hash(seed.as_bytes(), purpose.as_bytes())
}
```

क्लस्टर में `Keys::bind_node` चैलेंज कुंजी को `sibuna/challenge-node/v1` और नोड id पर उसी कुंजी के तहत कुंजी-युक्त BLAKE3 से बदल देता है, इसलिए चैलेंज को उसी नोड पर लौटना होगा जिसने उसे जारी किया, जबकि सत्र टोकन और आवश्यकता टिकट हर सदस्य पर वैध रहते हैं।

domain separation उद्देश्यों के बीच आकस्मिक पुनः उपयोग रोकता है; मास्टर सीड के लीक होने से हर व्युत्पन्न कुंजी लीक हो जाती है। क्लस्टर को केवल एक सीड पर सहमत होना होता है।

स्टेटलेस चैलेंज



चित्र 4 चैलेंज पहचानकर्ता और सत्र टोकन के वायर प्रारूप। हर रिकॉर्ड के ऊपर की संख्याएँ बाइट ऑफ़सेट हैं।

चैलेंज पहचानकर्ता एक स्वयं-प्रमाणित रिकॉर्ड है: संस्करण, एल्गोरिदम, कठिनाई, ओपनिंग की संख्या, जारी करने का समय, क्लाइंट का कुंजी-युक्त फ़िंगरप्रिंट, PRF से निकाला गया नॉनस, और उस नीति नियम का हैश जिसने चैलेंज माँगा, जिसके बाद चैलेंज कुंजी के तहत 16-बाइट टैग आता है। इसे जारी करने में कुछ भी नहीं लिखा जाता। सत्यापन रिकॉर्ड को डिकोड करता है, स्थिर समय में टैग जाँचता है, चैलेंज TTL के विरुद्ध आयु जाँचता है, सबमिट करने वाले क्लाइंट के विरुद्ध फ़िंगरप्रिंट जाँचता है, और उसके बाद ही प्रमाण की जाँच करता है।

लेम्मा 4 (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 लिए वैध प्रमाण स्वीकार हो चुका हो, और चैलेंज TTL के

बाद निकल जाता है। इसलिए $|S| \le r \cdot \mathrm{TTL}$, जब r उस अंतराल में स्वीकृत अलग-अलग समाधानों की दर को सीमित करता है। मुफ़्त चैलेंज लाने पर कोई टैग आवंटित नहीं होता। इम्प्लीमेंटेशन की क्षमता भी निश्चित है और भर जाने पर वह वैध समाधान को अस्वीकार कर सकता है। यह अवस्था की सीमा है, हर विरोधी की गणना-रणनीति पर निचली सीमा नहीं।

spent set 16-बाइट टैग पर Robin Hood open addressing के 16 शार्ड है (भाग V)। फ़िंगरप्रिंट क्लाइंट पते और User-Agent का कुंजी-युक्त हैश है, लंबाई से अलग किया गया ताकि कोई दो इनपुट जोड़ने (concatenation) से टकराएँ नहीं, और कुंजी-युक्त ताकि दूसरे क्लाइंट के फ़िंगरप्रिंट ऑफ़लाइन न निकाले जा सकें।

अभ्यास 3.3. टोकन पेलोड एक वर्क लेवल, `timestamp`, `expiry`, `rule_hash` और `fingerprint` रखता है। समझाएँ कि हर फ़ील्ड क्या रोकता है, यानी उसे हटाने पर क्या संभव हो जाता, और टैग को पाँचों को क्यों ढकना चाहिए।

अपरिवर्तनीय नियम समझाएँ. एक सहकर्मी सभी टोकन को Ed25519 पर ले जाने का प्रस्ताव देता है "क्योंकि वह अधिक मज़बूत है"। विश्वास क्षेत्र का तर्क, लागत का अंतर और पोस्ट-क्वांटम पहलू समझाएँ।

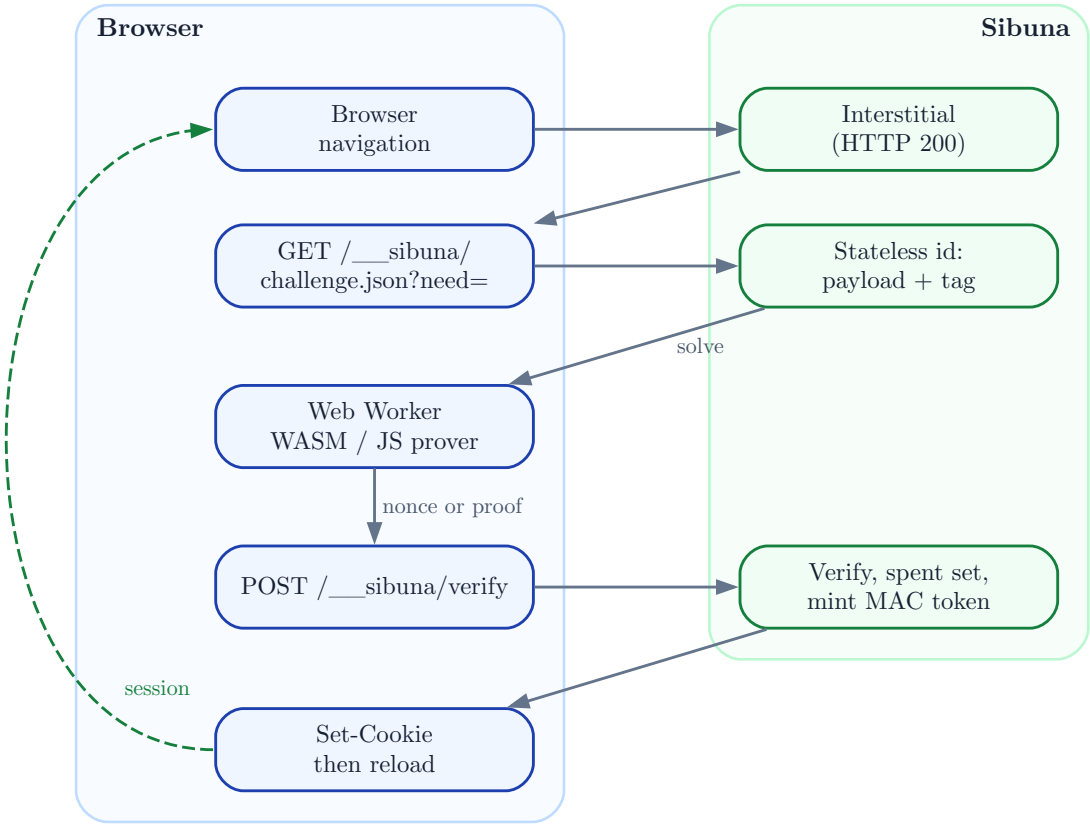
भाग IV

Sibuna प्रोटोकॉल

हम वायर प्रोटोकॉल को उसी रूप में निर्दिष्ट करते हैं जैसा वह लागू है: इंटरस्टिशियल का राउंड ट्रिप, स्टेटलेस चैलेंज रिकॉर्ड, समाधान के दोनों प्रारूप, सत्यापन का क्रम, एक-बार-उपयोग का प्रवर्तन, और सत्र कुकी।

चैलेंज का राउंड ट्रिप

उद्देश्य इस अध्याय के अंत तक हमें यह कर पाना चाहिए: किसी ब्राउज़र को उसके पहले अनुरोध से लेकर जारी किए गए सत्र तक ट्रेस करना, इसमें शामिल हर एंडपॉइंट और JSON फ़ील्ड का नाम बताना, और समझाना कि चैलेंज संरक्षित पाथ का नीति निर्णय अपने साथ क्यों रखता है।



चित्र 5 चैलेंज का राउंड ट्रिप। बाईं ओर के चरण ब्राउज़र में चलते हैं, और दाईं ओर के चरण Sibuna में।

1. वैध सत्र के बिना कोई नेविगेशन नीति इंजन तक पहुँचता है और `CHALLENGE` के रूप में वर्गीकृत होता है। चूँकि अनुरोध `text/html` स्वीकार करता है, डेमन इंटरस्टिशियल पेज और हेडर `X-Sibuna-Status: CHALLENGE` के साथ `200` उत्तर देता है; जो API क्लाइंट HTML स्वीकार नहीं करता, उसे JSON बॉडी के साथ `401` मिलता है, जिसका `challenge` URL लाने के लिए तैयार होता है। दोनों ही स्थितियों में रिस्पॉन्स एक आवश्यकता टिकट (requirement ticket) ले जाता है: इस निर्णय द्वारा माँगा गया चैलेंज (एल्गोरिदम, वर्क बिट्स, ओपनिंग) और नियम का हैश, जिसे इस क्लाइंट के लिए उसकी अपनी व्युत्पन्न कुंजी के तहत कुंजी-युक्त BLAKE3 टैग से सील किया जाता है, और जो एक चैलेंज अवधि तक वैध रहता है।
2. इंटरस्टिशियल `GET /__sibuna/challenge.json?path=<original URL>&need=<ticket>` लाता है। वैध टिकट चैलेंज तय करता है: कार्य उस निर्णय से निश्चित होता है जिसने उसे माँगा था, किसी रिपोर्ट किए गए URL और fetch के अपने हेडर से दोबारा नहीं निकाला जाता, जो नेविगेशन के हेडर (`Accept`, `Sec-Fetch-*`) से अलग होते हैं। वैध टिकट के बिना (कोई API क्लाइंट जो URL को अनदेखा करता है, या अवधि से आगे खुला छोड़ा गया पेज) सर्वर रिपोर्ट किए गए URL का मूल्यांकन करता है, जिसे उसी फ़ंक्शन से पाथ और क्वेरी में बाँटा जाता है जो अनुरोध पंक्ति पढ़ता है, और पाथ को उसी सामान्य रूप में रखा जाता है जिसका प्रवेश उपयोग करता है (SID 0009); 8 KiB से लंबा URL `414` के साथ अस्वीकार किया जाता है, कभी छोटा नहीं किया जाता, और जिस पाथ का सामान्य रूप न हो वह `400` के साथ। टिकट केवल सलाह देता है: प्रवेश फिर भी सत्र के वर्क लेवल की जाँच करता है, इसलिए रीप्ले किया गया या जाली टिकट केवल अपने धारक का कार्य बढ़ा सकता है।
3. रिस्पॉन्स चैलेंज रिकॉर्ड है:

   ```
   {"id":"AQEND...70
   chars","algorithm":"posw","difficulty":13,"challenges":16,"expires_at":1757241234}
   ```

   `hashcash` के लिए `difficulty` बिट की संख्या है और `challenges` का मान `0` है।
4. एक Web Worker इसे WebAssembly मॉड्यूल से हल करता है, या WebAssembly उपलब्ध न हो तो JavaScript प्रूवर से (भाग VII), और या तो यह पोस्ट करता है

   ```
   {"challenge_id":"...","nonce":"90766"}
   ```

   या यह

   ```
   {"challenge_id":"...","proof":"<base64url, 32(1 + t(n+1)) bytes>"}
   ```

   `POST /__sibuna/verify` पर।
5. सफलता पर सर्वर `Set-Cookie` हेडर के साथ `200 {"status":"ok"}` उत्तर देता है, और पेज अपना मूल स्थान दोबारा लोड करता है। बाद का हर अनुरोध कुकी ले जाता है और नीति मूल्यांकन से पहले ही प्रवेश पा जाता है।

पाथ चैलेंज के साथ क्यों जाता है इंटरस्टिशियल संरक्षित URL पर ही दिया जाता है, इसलिए ब्राउज़र जानता है कि वह कहाँ है। पाथ भेजने से सर्वर प्रति-नियम कठिनाई और एल्गोरिदम चुन पाता है (checkout रूट अनुक्रमिक कार्य के 20 वर्क बिट्स माँग सकता है जबकि कोई ब्लॉग Hashcash के 16 बिट माँगता है) और नियम हैश को टोकन में बाँध देता है, जहाँ `X-Sibuna-Rule-Hash` उसे अपस्ट्रीम तक रिपोर्ट करता है।

स्टेटलेस चैलेंज रिकॉर्ड

उद्देश्य चैलेंज एन्कोडर पढ़ें, हर फ़ील्ड का उद्देश्य समझें, और देखें कि हॉट पाथ पर entropy syscall के बिना प्रति-चैलेंज नॉनस कैसे निकाला जाता है।

इम्प्लीमेंटेशन: `Coordinator::create_challenge_with_spec` 36-बाइट पेलोड बनाता है, उसे चैलेंज कुंजी के तहत टैग करता है, और 52 बाइट को base64url में एन्कोड करके 70 अक्षरों का पहचानकर्ता बनाता है। `crates/challenge/src/coordinator.rs` में (क्रेट `sibuna-challenge`)।

```
pub fn create_challenge_with_spec(
    &self,
    client_ip: &[u8],
    user_agent: &[u8],
    now: u64,
    spec: ChallengeSpec,
    rule_hash: u64,
) -> Challenge {
    self.adaptive.observe(now.saturating_mul(1000));
    let effective = ChallengeSpec {
        difficulty: spec.difficulty.saturating_add(self.adaptive.bump()),
        ..spec
    };
    let fingerprint = self.fingerprint(client_ip, user_agent);
    let difficulty = effective.parameter();
    let challenges = match effective.algorithm {
        Algorithm::Hashcash => 0,
        Algorithm::Posw => effective.posw_challenges,
    };

    let mut payload = [0u8; ID_PAYLOAD_LEN];
    payload[0] = ID_VERSION;
    payload[1] = effective.algorithm.to_byte();
    payload[2] = difficulty;
    payload[3] = challenges;
    payload[4..12].copy_from_slice(&now.to_le_bytes());
    payload[12..20].copy_from_slice(&fingerprint.to_le_bytes());
    payload[20..28].copy_from_slice(&self.next_nonce(now,
fingerprint).to_le_bytes());
    payload[28..36].copy_from_slice(&rule_hash.to_le_bytes());
    let mut raw = [0u8; ID_RAW_LEN];
    raw[..ID_PAYLOAD_LEN].copy_from_slice(&payload);
    raw[ID_PAYLOAD_LEN..].copy_from_slice(&self.challenge_tag(&payload));

    Challenge {
        id: encoded::encode(&raw),
        algorithm: effective.algorithm,
        difficulty,
        challenges,
        expires_at: now.saturating_add(u64::from(self.challenge_ttl)),
    }
```

```
}
```

पहचानकर्ता एक `Encoded<70>` है: टेक्स्ट ठीक उसकी लंबाई वाली ऐरे में रहता है, इसलिए चैलेंज जारी करने में कभी आवंटन नहीं होता। `ChallengeSpec::parameter` वर्क बिट्स को टियर की अपनी इकाई में बदलता है, Hashcash बिट (अधिकतम 64) या PoSW गहराई (`difficulty - 3`, 4 से 24 के बीच सीमित)।
तीन विवरण ध्यान देने योग्य हैं:

- अनुकूली कठिनाई। कोऑर्डिनेटर तीन atomic में चैलेंज जारी होने की दर का exponentially weighted moving average रखता है। प्रति सेकंड 50 चैलेंज की आधार रेखा से ऊपर, दर के हर दोगुना होने पर एक वर्क बिट जुड़ता है, अधिकतम छह तक। बाढ़ घातीय रूप से अधिक चुकाती है, जबकि शांत ट्रैफ़िक आधार लागत पर ही रहता है।
- नॉनस एक PRF आउटपुट है, जिसे चैलेंज कुंजी से एक काउंटर, घड़ी और फ़िंगरप्रिंट पर गणना करके निकाला जाता है। यह अद्वितीय (काउंटर) और अप्रत्याशित (कुंजी) है, इसलिए विरोधी उन पहचानकर्ताओं के लिए पहले से समाधान नहीं निकाल सकता जो उसे जारी नहीं किए गए, और हॉट पाथ entropy के लिए कभी ऑपरेटिंग सिस्टम को कॉल नहीं करता।
- कठिनाई टैग के भीतर बँधी रहती है। क्लाइंट रिकॉर्ड संपादित करके कठिनाई कम नहीं कर सकता: किसी प्रमाण की जाँच से पहले ही टैग विफल हो जाएगा।

सत्यापन का क्रम और एक-बार उपयोग

उद्देश्य समझें कि सस्ती जाँचें प्रमाण से पहले क्यों चलती हैं, चैलेंज को केवल वैध प्रमाण के बाद ही spent क्यों चिह्नित किया जाता है, और एक ही समाधान के दो समवर्ती सबमिशन का निपटारा कैसे होता है।

इम्प्लीमेंटेशन: `Coordinator::verify_and_mint` पहचानकर्ता को डिकोड और प्रमाणित करता है, आयु और क्लाइंट बाइंडिंग जाँचता है, दर्ज टियर के लिए प्रमाण का सत्यापन करता है, टैग को spent के रूप में दर्ज करता है, और सत्र टोकन जारी करता है। `crates/challenge/src/coordinator.rs` में (क्रेट `sibuna-challenge`)।

```rust
pub fn verify_and_mint(
    &self,
    challenge_id: &[u8],
    solution: Solution<'_>,
    client_ip: &[u8],
    user_agent: &[u8],
    now: u64,
) -> Result<Verified, VerifyError> {
    let record = self.decode(challenge_id)?;
    let expires_at =
record.issued_at.saturating_add(u64::from(self.challenge_ttl));
    if now > expires_at || record.issued_at >
now.saturating_add(CLOCK_SKEW_SECONDS) {
        return Err(VerifyError::ChallengeExpired);
    }
    if record.fingerprint != self.fingerprint(client_ip, user_agent) {
        return Err(VerifyError::FingerprintMismatch);
    }
    check_solution(challenge_id, &record, solution)?;
```

```
    self.spent.mark_spent(&record.tag, expires_at, now).map_err(|error| match
error {
        SpendError::DoubleSpend => VerifyError::DoubleSpendAttempt,
        SpendError::Full => VerifyError::StoreFull,
    })?;
    let work = WorkLevel {
        algorithm: record.algorithm.to_byte(),
        bits: record.algorithm.work_bits(record.difficulty),
    };
    Ok(Verified {
        token: self.mint_token(now, record.rule_hash, record.fingerprint, work),
        algorithm: record.algorithm,
        difficulty: record.difficulty,
        challenges: record.challenges,
        ttl_seconds: self.token_ttl,
    })
}
```

यह क्रम जानबूझकर है। टैग, आयु और बाइंडिंग स्थिर-समय जाँचें हैं जो किसी भी हैशिंग से पहले कचरे को अस्वीकार कर देती हैं। गलत प्रमाण को चैलेंज का उपभोग नहीं करना चाहिए, इसलिए spent set में तभी लिखा जाता है जब `check_solution` सफल हो। एक ही वैध समाधान का सत्यापन करने वाले दो थ्रेड `mark_spent` पर होड़ करते हैं; शार्ड लॉक उन्हें क्रम में लगाता है, ठीक एक जीतता है, और दूसरे को `VerifyError::DoubleSpendAttempt` मिलता है। शर्त `issued_at > now + 60` (`CLOCK_SKEW_SECONDS` का मान 60 है) उन रिकॉर्ड को अस्वीकार करती है जिन्हें ऐसे नोड ने जारी किया जिसकी घड़ी एक मिनट से अधिक आगे चल रही है, क्योंकि अन्यथा वे किसी चैलेंज की अवधि बढ़ा देते। त्रुटियाँ क्लाइंट तक Elm-शैली के निदान के साथ `400` के रूप में पहुँचती हैं (भाग X): हर `VerifyError` एक `sibuna_core::errors::ErrorKind` में बदलता है, जिसकी व्याख्या रिस्पॉन्स बॉडी होती है। end-to-end परीक्षण रिस्पॉन्स बॉडी में `DOUBLE SPEND`, `FINGERPRINT MISMATCH`, `WRONG SOLUTION TYPE` और `MALFORMED CHALLENGE` की पुष्टि करते हैं।

सत्र कुकी

उद्देश्य कुंजी-युक्त हैश वाला टोकन पढ़ें, कुकी के एट्रिब्यूट समझें, और क्लाइंट बाइंडिंग समझाएँ।

इम्प्लीमेंटेशन: `token::mac::mint` और `token::mac::verify` 40-बाइट पेलोड को सीरियलाइज़ करता है, उस पर कुंजी-युक्त BLAKE3 के पहले 16 बाइट जोड़ता है, और स्थिर-समय तुलना से सत्यापन करता है। `crates/challenge/src/token/mac.rs` में (क्रेट `sibuna-challenge`)। `Coordinator::verify_cookie` इसे क्लाइंट के फ़िंगरप्रिंट के साथ कॉल करता है।

```
pub fn verify(
    key: &[u8; 32],
    text: &[u8],
    now: u64,
    expected_fingerprint: Option<u64>,
) -> Result<Payload, TokenError> {
    if text.len() != TOKEN_LEN {
```

```
        return Err(TokenError::InvalidTokenLength);
    }
    let raw: [u8; RAW_LEN] =
        encoded::decode::<RAW_LEN,
TOKEN_LEN>(text).ok_or(TokenError::InvalidEncoding)?;
    let (payload, given) = raw.split_at(PAYLOAD_LEN);
    let mut bytes = [0u8; PAYLOAD_LEN];
    bytes.copy_from_slice(payload);
    // स्थिर समय: जल्दी बाहर निकलना जालसाज़ को बता देता कि टैग के कितने बाइट सही थे।
    if !ct::eq(&tag(key, &bytes), given) {
        return Err(TokenError::InvalidTokenSignature);
    }
    let payload = Payload::from_bytes(&bytes)?;
    payload.check(now, expected_fingerprint)?;
    Ok(payload)
}
```

पेलोड पार्स होने से पहले टैग जाँचा जाता है: किसी दूसरे संस्करण वाला पेलोड `TokenError::InvalidEncoding` के साथ तभी अस्वीकार होता है जब यह ज्ञात हो जाए कि वह प्रामाणिक है, और जाली पेलोड कभी पार्सर तक नहीं पहुँचता। कुकी इस रूप में भेजी जाती है

```
Set-Cookie: __sibuna_token=<75 chars>; Path=/; Max-Age=86400; HttpOnly;
    SameSite=Lax[; Secure]
```

`HttpOnly` इसे पेज स्क्रिप्ट से दूर रखता है, `SameSite=Lax` शीर्ष-स्तर के नेविगेशन की अनुमति देते हुए cross-site रीप्ले रोकता है, और जब TLS डेमन के सामने टर्मिनेट होता है तब `--secure-cookie` के साथ `Secure` जोड़ा जाता है। चूँकि पेलोड पते और User-Agent का कुंजी-युक्त फ़िंगरप्रिंट रखता है, किसी दूसरी मशीन पर कॉपी की गई कुकी `TokenError::TokenBoundAddressMismatch` के साथ विफल होती है; end-to-end परीक्षण "the cookie is bound to the client identity" ठीक इसी पाथ को परखता है।

पेलोड एक संस्करण बाइट और धारक द्वारा चुकाए गए वर्क लेवल से शुरू होता है: तंत्र (Hashcash या PoSW) और वास्तव में हल किए गए वर्क बिट्स, लोड के अनुसार हुई किसी भी बढ़ोतरी सहित। कोई सत्र चैलेंज को तभी पार करता है जब उसका लेवल रूट की माँग तक पहुँचता है, जिसकी गणना नियम की कठिनाई से उसी सीमांकन (clamping) के ज़रिए होती है जो चैलेंज स्वयं लागू करता है। लेवल क्रमबद्ध हैं, नामित नहीं: 24 बिट पर अर्जित कुकी हर 16-बिट रूट को भी ढकती है, जबकि 24-बिट रूट पर प्रस्तुत 16-बिट कुकी अधिक कठिन इंटरस्टिशियल तक पहुँचती है और नई कुकी पुरानी को बदल देती है। `rule_hash` उस नियम की ऑडिट पहचान बना रहता है जिसने चैलेंज जारी किया; इसे अपस्ट्रीम तक रिपोर्ट किया जाता है, प्रवेश के लिए कभी उपयोग नहीं किया जाता। WAF के निष्कर्षों और स्पष्ट अस्वीकृतियों को सत्र कभी पार नहीं करता। end-to-end परीक्षण "a session earned on a cheaper route does not admit a route that demands more work" दोनों दिशाओं को परखता है।

> **फ़ॉरवर्ड किए गए पते** इनग्रेस के पीछे क्लाइंट का पता `X-Forwarded-For` में आता है। Sibuna इस हेडर पर केवल `--trust-forwarded` के साथ भरोसा करता है (forward-auth मोड में यह अपने आप लागू होता है)। सीधे इंटरनेट के सामने खुले रिवर्स प्रॉक्सी को इसे बंद रखना चाहिए, अन्यथा कोई भी क्लाइंट अपनी बाइंडिंग और अपना रेट-लिमिट बकेट स्वयं चुन सकता है।

अभ्यास 4.1. एक NAT पते के पीछे समान User-Agent वाले दो ब्राउज़र के अनुक्रम का खाका बनाएँ। वे कौन-सी जाँचें साझा करते हैं, कौन-सी नहीं, और उस NAT पर मौजूद हमलावर को दूसरे ब्राउज़र की कुकी दोबारा उपयोग करने के लिए क्या चाहिए?

अपरिवर्तनीय नियम समझाएँ. किसी ऑपरेटर को समझाएँ कि डेमन लंबित चैलेंज को "याद" क्यों नहीं रखता, और यह कोई शॉर्टकट नहीं बल्कि सुरक्षा की विशेषता क्यों है।

भाग V

Rust में शून्य-आवंटन इंजीनियरिंग

हम एक कनेक्शन को *accept* से रिस्पॉन्स तक एक निश्चित बफ़र के ज़रिए देखते हैं, और उन समवर्ती डेटा संरचनाओं की जाँच करते हैं जो कई कनेक्शन थ्रेड को अनुरोध पाथ पर हीप आवंटन के बिना अवस्था साझा करने देती हैं। जहाँ मेमोरी सुरक्षा के लिए अन्यथा अपनी अलग व्यवस्था *(reader count, arena, descriptor table, teardown chain)* चाहिए होती, वहाँ हम दिखाते हैं कि *Rust* की ओनरशिप उसकी जगह यह कैसे देती है।

कनेक्शन लूप

उद्देश्य इस अध्याय के अंत तक हमें यह समझा पाना चाहिए कि एक 64 KiB बफ़र पूरे keep-alive कनेक्शन की सेवा कैसे करता है, हेड और बॉडी को कॉपी किए बिना कैसे खोजा जाता है, हर कनेक्शन को अपना सीमित थ्रेड क्यों मिलता है, प्रॉक्सी ऑरिजिन की फ़्रेमिंग कैसे जानता है ताकि प्रॉक्सी किया गया क्लाइंट कनेक्शन खुला रह सके, और इनमें से कौन-सी गारंटी borrow checker हमारे लिए सिद्ध करता है।

प्रति कनेक्शन एक थ्रेड, सीमित

Accept वर्कर (`--workers`, डिफ़ॉल्ट रूप से प्रति CPU एक, अधिकतम 64) listening सॉकेट साझा करते हैं। हर स्वीकृत कनेक्शन एक मेगाबाइट स्टैक वाले उसके अपने थ्रेड को सौंप दिया जाता है, और accept लूप सीधे `accept` पर लौट जाता है। जो थ्रेड अपना कनेक्शन पूरा कर लेता है, वह रुककर अगला कनेक्शन लेता है, क्योंकि हर कनेक्शन के लिए थ्रेड बनाने और समाप्त करने से दूसरे थ्रेड अटक जाते थे (भाग VIII, SID 0002)। विकल्प, यानी कनेक्शन को accept थ्रेड पर ही पूरा सर्व करना, पहला डिज़ाइन था: एक क्लाइंट पर इसके माप अच्छे थे और चौंसठ पर बुरे, क्योंकि उनमें से साठ किसी वर्कर के 256 अनुरोधों का कोटा पूरा होने की प्रतीक्षा करते थे, और हर प्रतीक्षारत क्लाइंट के साथ टेल लेटेंसी बढ़ती जाती थी। प्रति कनेक्शन एक थ्रेड के साथ, 64 कनेक्शन पर p99 पूरे उत्पाद के हर वर्कलोड में एक मिलीसेकंड से कम रहता है, 305 us से 857 us तक (`benchmarks/results/product-latest.md`; भाग VIII)।
कनेक्शन थ्रेड की संख्या `--max-connections` से सीमित है (डिफ़ॉल्ट 1,024, अधिकतम 8,192)। सीमा पार होने पर accept लूप नए सॉकेट पर `503 Service Unavailable` उत्तर देता है और कुछ भी शुरू किए बिना उसे बंद कर देता है, और इस घटना को `sibuna_overloaded_total` में गिनता है।

इम्प्लीमेंटेशन: `accept_loop` कनेक्शन स्वीकार करता है, हर एक को कनेक्शन सीमा के विरुद्ध जाँचता है, और उसे task pool से उसके अपने थ्रेड पर शुरू करता है। `cmd/sibuna/src/server.rs` में (क्रेट `sibuna`)।

```
fn accept_loop(server: &Arc<Server>, listener: &TcpListener) {
    while !server.is_stopping() {
        let stream = match listener.accept() {
            Ok((stream, _)) => stream,
            // ... descriptor या मेमोरी खत्म: spin करने के बजाय 1 ms रुकें
        };
        let limit = server.max_connections.load(Ordering::Relaxed);
```

```
        if server.connections.fetch_add(1, Ordering::Relaxed) >= limit {
            server.connections.fetch_sub(1, Ordering::Relaxed);
            server.metrics.overloaded.bump();
            let text = "Service Unavailable: connection limit reached";
            reject(server, &stream, text);
            continue;
        }
        set_no_delay(&stream);
        let task_server = Arc::clone(server);
        let launched = server.tasks.launch(move || {
            let _counted = Counted(&task_server);
            crate::connection::serve(&task_server, &stream);
        });
        // ... launch विफल होने पर स्थान लौटता है और एक ओवरलोड गिना जाता है
    }
}
```

`Counted` एक guard है जिसका `Drop` सीमा में कनेक्शन का स्थान वापस कर देता है, थ्रेड चाहे जैसे समाप्त हो, return से या panic से। `503` वाला पाथ सॉकेट की अपनी ओर को बंद करने से पहले वह सब पढ़कर खाली कर देता है जो क्लाइंट पहले ही भेज चुका है: बिना पढ़े बाइट वाले सॉकेट को बंद करने पर reset उत्तर जाता है, और reset `503` को मिटा देगा। थ्रेड एक `TaskPool` (`cmd/sibuna/src/tasks.rs`) से आते हैं, जो join किए जा सकने वाले स्लॉट की एक निश्चित टेबल है। समाप्त हुआ थ्रेड join होने तक अपना स्टैक रखता है, इसलिए स्लॉट को तब join किया जाता है जब उसे दोबारा लिया जाता है, और नया task कर्सर आगे बढ़ाने के बजाय सबसे निचला समाप्त स्लॉट लेता है। कर्सर किसी समाप्त स्लॉट तक टेबल का पूरा चक्कर लगाने के बाद ही दोबारा पहुँचता, जिससे हज़ारों समाप्त थ्रेड मेमोरी में बने रहते, और मेमोरी खुले कनेक्शनों के बजाय सर्व किए गए कनेक्शनों के साथ बढ़ती। दूसरा guard, `Finished`, task समाप्त होने पर स्लॉट को मुक्त चिह्नित करता है, इसलिए panic करने वाला कनेक्शन भी अपना स्लॉट मुक्त कर देता है।

मेमोरी भी इसी तरह सीमित है। कनेक्शन थ्रेड अपने स्टैक पर अपना 64 KiB रीड बफ़र और 16 KiB राइट बफ़र रखता है, और हीप पर 40 KiB का प्रॉक्सी वर्कस्पेस, जो कनेक्शन शुरू होने पर एक बार आवंटित होता है। इसलिए सबसे खराब स्थिति एक ज्ञात संख्या है, यह इस पर निर्भर नहीं कि कोई क्लाइंट कितने सॉकेट खोल सकता है। आइडल रीपर (नीचे) उन कनेक्शनों को बंद करता है जो भेजना बंद कर देते हैं, इसलिए कोई धीमा क्लाइंट किसी थ्रेड को `--idle-timeout` से आगे नहीं रोक सकता।

आवंटक का चयन

Rust 1.100 में `Allocator` API का मुख्य भाग और `Vec` तथा `Box` के आवंटक पैरामीटर स्थिर हो गए हैं (रिलीज़ नोट)। हम प्रक्रिया का आवंटक बदले बिना किसी एक संग्रह का आवंटक चुन सकते हैं। इस API से अनुरोध पथ पर आवंटन की और बचत नहीं होती: तैयार कनेक्शन पर अनुरोध पहले से कोई मेमोरी आवंटित नहीं करते, जिसकी जाँच `cmd/sibuna/tests/steady_state.rs` और `crates/net/tests/steady_state.rs` करते हैं। प्रॉक्सी हर कनेक्शन के लिए अपने तीन बफ़र एक बार आवंटित करता है और उनका पुनः उपयोग करता है। सीमक और प्रयुक्त प्रमाणों का सेट भी अपनी तालिकाएँ एक ही बार आवंटित करते हैं; शून्य से आरंभ किए गए आवंटन में अछुए पृष्ठ निवासी मेमोरी नहीं लेते, जिसकी जाँच `crates/store/tests/resident.rs` करता है।

आवंटक में बदलाव की जाँच करते समय हम पहले नीति स्नैपशॉट के निर्माण और कंसोल के JSON पार्सिंग को मापेंगे। उनके अस्थायी संग्रह अनुरोध पथ के बाहर मेमोरी आवंटित करते हैं, लेकिन किसी संग्रह का आवंटक उसके तत्वों के आवंटन को अपने आप नियंत्रित नहीं करता। हमारा JSON वृक्ष एस्केप वाली स्ट्रिंग का स्वामित्व भी `Cow<str>` के माध्यम से रखता है, जिसका स्वामित्व वाला रूप `String` है; Rust 1.100 में `String` का कोई आवंटक पैरामीटर नहीं है। इस रिलीज़ में `Arc::new_in` भी अस्थिर है (API दस्तावेज़)। प्रकाशित इंजन `Arc` के माध्यम से तब तक साझा रहते हैं जब तक

अंतिम पाठक उन्हें छोड़ नहीं देता, इसलिए पुनर्निर्माण उस मेमोरी को रीसेट नहीं कर सकता जिसे पुराना इंजन अभी उपयोग कर रहा है। किसी अरीना को इन जीवनकालों और पार्सर की इनपुट सीमाओं को बनाए रखना होगा।

हम कंसोल के WebAssembly मॉड्यूल के मौजूदा वैश्विक आवंटक `WipeOnFree` को `cmd/console-ui/src/abi.rs` में बनाए रखते हैं। यह मुक्त किए गए ब्लॉक मिटाता है, जिनमें पुनः आवंटन के बाद पुराना ब्लॉक भी शामिल है। किसी संग्रह के लिए सीधे `System` चुनने से यह मिटाना छूट जाएगा। कार्य प्रमाण के WebAssembly मॉड्यूल में कोई हीप आवंटक नहीं है, और Argon2 पहले से कॉलर के स्वामित्व वाले कार्यक्षेत्र का पुनः उपयोग करता है। इसलिए हम मापे गए कार्यभार से बदलाव उचित सिद्ध होने तक मौजूदा आवंटक और Rust 1.94 संगतता बनाए रखते हैं। किसी विकल्प को मिटाना, सीमाबद्ध इनपुट प्रसंस्करण और आवंटन रहित अनुरोध पथ बनाए रखते हुए विलंबता या निवासी मेमोरी में सुधार करना होगा।

प्रति कनेक्शन एक बफ़र

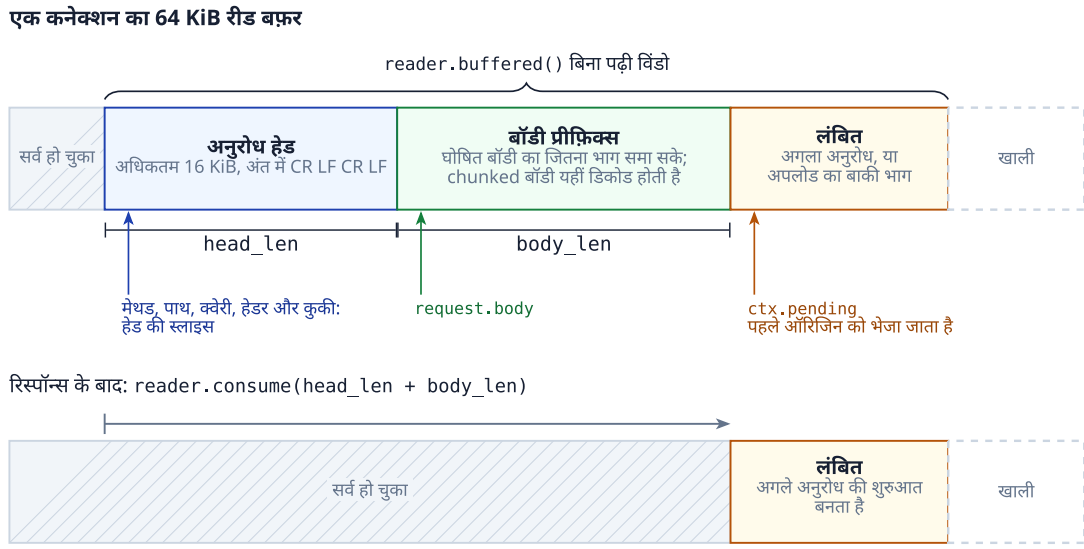
अनुरोध पाथ पर हर डायनामिक आवंटन denial-of-service का एक लीवर है: दिनों में बढ़ता विखंडन (fragmentation), थ्रेड के बीच एलोकेटर लॉक की होड़, और उन हमलावरों द्वारा प्रवर्धन जो आवंटन को अधिकतम करने वाले अनुरोध गढ़ते हैं। Sibuna का नियम पूर्ण है: वर्गीकरण, सत्यापन और प्रॉक्सी को सौंपना शून्य हीप आवंटन के साथ चलते हैं। कनेक्शन हैंडलर सॉकेट के पूरे जीवनकाल के लिए एक 64 KiB रीड बफ़र और एक 16 KiB राइट बफ़र का स्वामी है; बाकी सब उनमें स्लाइस है।

> **इम्प्लीमेंटेशन: `connection::serve`** कनेक्शन को आइडल रीपर के साथ पंजीकृत करता है, बफ़र, प्रॉक्सी वर्कस्पेस और इंजन cache को एक बार तैयार करता है, और अधिकतम 4,096 अनुरोध सर्व करता है। `cmd/sibuna/src/connection.rs` में (क्रेट `sibuna`)।

```rust
pub fn serve(server: &Server, stream: &TcpStream) {
    let Some(lease) = server.idle.register(stream) else { return };
    // ...
    let mut read_buffer = [0u8; MAX_REQUEST_BYTES]; // 64 KiB
    let mut write_buffer = [0u8; WRITE_BUFFER]; // 16 KiB
    let mut reader = Reader::new(stream, &mut read_buffer[..]);
    let mut peer = PeerText::default();
    peer.format(stream.peer_addr().ok());
    let mut link = Link {
        server,
        stream,
        writer: Writer::new(stream, &mut write_buffer[..]),
        workspace: Workspace::new(),
        cache: EngineCache::new(),
        activity: lease.activity(),
        last: false,
    };
    for served in 0..MAX_REQUESTS_PER_CONNECTION {
        link.last = served + 1 == MAX_REQUESTS_PER_CONNECTION;
        if !serve_one(&mut reader, &mut link, peer.as_bytes()) {
            break;
        }
        lease.touch();
    }
}
```

`read_head` बफ़र को तब तक भरता है जब तक खाली पंक्ति न आ जाए, और केवल पिछली खोज के बाद आए बाइट में खोजता है, और 16 KiB से बड़े हेड को `431` के साथ अस्वीकार करता है। `parse_request` (क्रेट `sibuna-http`) एक

`Request<'a>` बनाता है जिसके मेथड, पाथ, क्वेरी, हेडर और कुकी बफ़र की स्लाइस हैं। घोषित बॉडी को जितनी समा सके उतनी भरा जाता है और हेड के बाद स्लाइस किया जाता है। फिर `reader.consume(head_len + body_len)` रीडर को आगे बढ़ाता है, ताकि अगला keep-alive अनुरोध साफ़ शुरू हो; रीडर बिना पढ़े बाइट को आगे केवल तब खिसकाता है जब उसे जगह चाहिए। chunked बॉडी को हेड के पीछे उसी स्थान पर डिकोड किया जाता है (SID 0009): हर चरण की फ़्रेमिंग बफ़र से काट दी जाती है, इसलिए डिकोड किया गया प्रीफ़िक्स निरीक्षण के लिए सटा हुआ रहता है। आंतरिक रूट लंबाई से सीमांकित हैं और कनेक्शन खुला रखते हैं। प्रॉक्सी किए गए अनुरोध भी इसे खुला रखते हैं, बशर्ते ऑरिजिन का रिस्पॉन्स फ़्रेम किया गया हो और बॉडी पूरी तरह बफ़र हुई हो; अगले अनुभाग दिखाते हैं कि प्रॉक्सी यह कैसे तय करता है।



चित्र 6 एक अनुरोध सर्व होते समय कनेक्शन का रीड बफ़र। अनुरोध का हर फ़ील्ड इसकी एक स्लाइस है।

```rust
fn serve_one<W: Write>(
    reader: &mut Reader<'_, &TcpStream>,
    link: &mut Link<'_, W>,
    peer: &[u8],
) -> bool {
    let head_len = match read_head(reader) {
        Head::Complete(length) => length,
        Head::TooLarge => { /* 431 "Request head exceeds 16 KB" */ return false; }
        Head::Closed | Head::Failed => return false,
    };
    let framing = {
        let bytes = reader.buffered();
        let head = bytes.get(..head_len).unwrap_or_default();
        let request = match parse_request(head) {
            Ok(request) => request,
            Err(error) => return reject_head(link, error),
        };
        let declared = request.content_length().unwrap_or(0);
        // ... Expect: 100-continue
        if declared == 0 && !request.chunked {
            let upload = Upload::declared(&request);
            let pending = bytes.get(head_len..).unwrap_or_default();
            let keep = dispatch(link, request, pending, upload, 0, peer);
```

```
            reader.consume(head_len);
            return keep;
        }
        (declared, request.chunked)
    };
    serve_with_body(reader, link, peer, head_len, framing)
}
```

borrow checker क्या सिद्ध करता है अनुरोध की स्लाइस केवल बफ़र के अगली बार भरे जाने तक वैध रहती हैं। Sibuna इस नियम को टिप्पणी के रूप में नहीं, एक प्रकार (type) के रूप में बताता है: अनुरोध एक `Request<'a>` है, और `'a` रीडर के बफ़र का साझा borrow है। बफ़र भरने या consume करने के लिए `&mut reader` चाहिए, जिसे कंपाइलर तब तक अस्वीकार करता है जब तक उससे उधार लिया गया `Request` अभी उपयोग में है। `serve_one` में अनुरोध को `dispatch` में move किया जाता है, और उस कॉल के लौटने के बाद ही `reader.consume` चल सकता है; इन्हें उलटे क्रम में लिखने पर कोड कंपाइल नहीं होता।

यही नियम दो ऐसे चुनावों को समझाता है जो पहली नज़र में अजीब लगते हैं। पहला, `serve_with_body` हेड को दूसरी बार पार्स करता है। पहले पार्स को बॉडी बफ़र होने से पहले अपना borrow समाप्त करना होगा, क्योंकि chunked बॉडी को बफ़र करने से बफ़र के भीतर बाइट खिसकते हैं; दूसरा पार्स स्थिर हो चुके बफ़र को उधार लेता है। पार्सिंग नियतात्मक है और कुकी वाले ब्राउज़र अनुरोध के लिए लगभग 133 ns लेती है (`benchmarks/results/primitives-latest.md` में `zero_copy_request_and_cookie`, 132.5 ns), इसलिए दूसरे पार्स की लागत कम है और वह वही निर्णय देता है। दूसरा, प्रॉक्सी पार्स किए गए अनुरोध और लंबित बाइट को उधार लेता है और उसे कनेक्शन बफ़र दोबारा नहीं भरना चाहिए, इसलिए वह क्लाइंट के आगे के बाइट अपने अलग अपलोड बफ़र में पढ़ता है। चूँकि वह पढ़ा हुआ वापस नहीं लौटा सकता, क्लाइंट कनेक्शन केवल तभी रखा जाता है जब बॉडी पूरी तरह बफ़र हुई हो।

प्रॉक्सी हेड का पुनर्लेखन

हेड को ज्यों का त्यों फ़ॉरवर्ड नहीं किया जाता। `write_request_head` (`crates/net/src/request.rs`, क्रेट `sibuna-net`) अनुरोध पंक्ति और हर हेडर को ऑरिजिन writer में दोबारा लिखता है, और hop-by-hop फ़ील्ड (`Connection`, `Keep-Alive`, `Proxy-Connection`, `Transfer-Encoding`, `TE`, `Trailer`, `Upgrade`, प्रॉक्सी प्राधिकरण फ़ील्ड), कोई भी आने वाला फ़ॉरवर्डिंग या ऑडिट फ़ील्ड (`X-Forwarded-For`, `X-Real-IP`, `Forwarded`, `X-Forwarded-Proto`, `X-Forwarded-Host`, `X-Forwarded-Port`, `X-Sibuna-Status`, `X-Sibuna-Rule`), `Expect`, `Content-Length`, और हर वह फ़ील्ड जिसे अनुरोध का `Connection` हेडर नामित करता है, हटा देता है। फिर वह ऑडिट समुच्चय जोड़ता है। बॉडी की फ़्रेमिंग बनाई जाती है, कभी कॉपी नहीं की जाती:

```
for header in request.headers() {
    if is_hop_by_hop(header.name) || nominated(request, header.name) {
        continue;
    }
    out.write_all(header.name)?;
    out.write_all(b": ")?;
    out.write_all(header.value)?;
    out.write_all(b"\r\n")?;
}
match upload {
    Upload::None => {}
    Upload::Length(length) => {
```

```
        out.write_all(b"Content-Length: ")?;
        write_decimal(out, *length)?;
        out.write_all(b"\r\n")?;
    }
    Upload::Chunked(_) => out.write_all(b"Transfer-Encoding: chunked\r\n")?,
}
```

`Content-Length` दोबारा लिखा जाता है। जो chunked बॉडी कनेक्शन बफ़र के भीतर समाप्त हो गई, उसे उसी स्थान पर डिकोड करके उसकी लंबाई के साथ भेजा जाता है। लंबी बॉडी को `Transfer-Encoding: chunked` के रूप में घोषित किया जाता है और एक बार में एक रीड करके दोबारा chunk किया जाता है (SID 0009): हर रीड प्रॉक्सी के 16 KiB अपलोड बफ़र में आती है, वहीं डिकोड होती है, और एक canonical chunk के रूप में बाहर जाती है, इसलिए ऑरिजिन कभी क्लाइंट के chunk आकार, extension या trailer नहीं देखता। ऑडिट समुच्चय:

```
Connection: keep-alive
X-Forwarded-For: <client>
X-Forwarded-Proto: <scheme>
X-Real-IP: <client>
X-Sibuna-Status: PASS
X-Sibuna-Rule: session | robots-txt | ip/cidr-trie | ...
```

बफ़र से बड़ी बॉडी को रिस्पॉन्स वापस स्ट्रीम होने से पहले क्लाइंट से ऑरिजिन writer तक 16 KiB के टुकड़ों में रिले किया जाता है (`relay_body`)। `crates/net/src/request.rs` में यूनिट परीक्षण `proxy_head_rewrite_drops_hop_by_hop_headers_and_injects_audit_fields` पुष्टि करता है कि पुनर्लेखन एक जाली `X-Forwarded-For` को हटा देता है और ऑडिट फ़ील्ड जोड़ता है।
प्रॉक्सी के तीन बफ़र, ऑरिजिन रीड बफ़र (एक रिस्पॉन्स हेड, 16 KiB), ऑरिजिन राइट बफ़र (8 KiB) और अपलोड बफ़र (16 KiB), एक `Workspace` में रहते हैं जिसे कनेक्शन एक बार आवंटित करता है और हर अनुरोध के लिए दोबारा उपयोग करता है। वर्कस्पेस उनके जीवनकाल को एक्सचेंज का नहीं, कनेक्शन का बना देता है, और ऑरिजिन रीडर और writer उन्हें एक समय पर एक एक्सचेंज के लिए उधार लेते हैं।

ऑरिजिन रिस्पॉन्स का रिले

पहला प्रॉक्सी ऑरिजिन के बंद होने तक उसके बाइट स्ट्रीम करता था और फिर क्लाइंट को बंद कर देता था: बिना पार्सिंग के सही फ़्रेमिंग, लेकिन प्रति प्रॉक्सी अनुरोध एक नए TCP कनेक्शन की कीमत पर। लोड जनरेटर के तहत यह कीमत `TIME_WAIT` में दसियों हज़ार सॉकेट के रूप में दिखी, और loopback पर ephemeral पोर्ट समाप्त हो गए। अब प्रॉक्सी ऑरिजिन का हेड (अधिकतम 16 KiB) पढ़ता है और RFC 9112 के नियमों से बॉडी को वर्गीकृत करता है:

> **इम्प्लीमेंटेशन: `parse_response_head`** स्टेटस, चार फ़्रेमिंग में से एक (HEAD, 1xx, 204 और 304 के लिए कोई बॉडी नहीं; एक `Content-Length`; chunked transfer coding; या close-delimited) और यह लौटाता है कि ऑरिजिन कनेक्शन रखता है या नहीं। `crates/net/src/response.rs` में (क्रेट `sibuna-net`)।

```
pub enum Framing { None, Length(u64), Chunked, UntilClose }

pub fn parse_response_head(
    head: &[u8],
    head_request: bool,
) -> Option<ResponseHead> {
    let status = status_code(head)?;
```

```
    let mut fields = FramingFields {
        keep_alive: head.get(7) == Some(&b'1'),
        ..Default::default()
    };
    for line in crlf_lines(head).skip(1) {
        if line.is_empty() {
            break;
        }
        fields.read_line(line)?; // Transfer-Encoding, Content-Length, Connection
    }
    let mut framing = fields.length.map_or(Framing::UntilClose, Framing::Length);
    if head_request || status / 100 == 1 || status == 204 || status == 304 {
        framing = Framing::None;
    } else if fields.encoded {
        framing =
            if fields.chunked { Framing::Chunked } else { Framing::UntilClose };
    }
    let mut keep_alive = fields.keep_alive;
    if framing == Framing::UntilClose || fields.closing {
        keep_alive = false;
    }
    if fields.nominates_framing {
        return None;
    }
    Some(ResponseHead { status, framing, keep_alive })
}
```

पार्सर वह अस्वीकार करता है जिसे प्रॉक्सी को रिले नहीं करना चाहिए: खराब स्टेटस पंक्ति, ऐसा फ़ील्ड नाम जो टोकन नहीं है, मान में कंट्रोल बाइट, परस्पर विरोधी लंबाइयाँ, दोहराई गई या गैर-अंतिम `chunked` coding, और ऐसा `Connection` फ़ील्ड जो `Content-Length` या `Transfer-Encoding` को नामित करता है। इसका कार्य हेड में रैखिक है, और यह कभी panic नहीं करता।

हेड क्लाइंट को दोबारा लिखा जाता है, जिसमें ऑरिजिन के `Connection` हेडर की जगह Sibuna का अपना निर्णय होता है, और बॉडी ठीक-ठीक रिले की जाती है: लंबाई के लिए `copy_exact`, chunked coding के लिए chunk दर chunk (आकार पंक्ति, डेटा, trailer), और पुरानी (legacy) स्थिति के लिए `copy_remaining`। केवल legacy स्थिति क्लाइंट को बंद करती है। हर रिले एक ही कॉपी चरण पर बना है: स्रोत के पास जो पहले से है उसे आगे भेजें, और केवल तब, जब उसके पास कुछ न हो, गंतव्य को flush करें और फिर एक बार पढ़ें। इसलिए छोटे राइट की धारा जैसे-जैसे बनती है वैसे-वैसे पहुँचाई जाती है, जबकि बड़ा स्थानांतरण फिर भी प्रति रीड और राइट एक पूरा बफ़र ले जाता है।

```
let keep = client_keep_alive && parsed.framing != Framing::UntilClose;
write_client_head(client, head, keep).map_err(client_head_error)?;
origin.consume(head_length);
match parsed.framing {
    Framing::None => {}
    Framing::Length(length) => copy_exact(origin, client, length, progress)?,
    Framing::Chunked => relay_chunked(origin, client, progress)?,
    Framing::UntilClose => copy_remaining(origin, client, progress)?,
}
client.flush().map_err(|_| ProxyError::ClientFailed)?;
Ok(Relayed { client_keep: keep, origin_reusable: parsed.keep_alive })
```

आठ फ़ील्ड वाले ऑरिजिन हेड को पार्स करने में 322 ns लगते हैं, और loopback पर प्रॉक्सी किए गए 1 KiB रिस्पॉन्स वाले keep-alive `GET` में 29.8 us लगते हैं (अक्टूबर 2026 में मापा गया; `crates/net/src/lib.rs` मापन का दस्तावेज़ देता है)।

ऑरिजिन पूल

फ़्रेमिंग पार्स करने से प्रॉक्सी को यह भी पता चलता है कि ऑरिजिन सॉकेट दोबारा उपयोग हो सकता है या नहीं: `Connection: close` के बिना HTTP/1.1 रिस्पॉन्स (या `keep-alive` के साथ HTTP/1.0 रिस्पॉन्स), जिसकी बॉडी पूरी तरह consume हो चुकी हो, सॉकेट को एक साफ़ अनुरोध सीमा पर छोड़ता है। ऐसे सॉकेट 256 निष्क्रिय ऑरिजिन कनेक्शनों के एक निश्चित पूल में जाते हैं, और अगला प्रॉक्सी अनुरोध नया कनेक्शन बनाने के बजाय उनमें से एक लेता है। पूल के बिना हर प्रॉक्सी अनुरोध एक नया ऑरिजिन कनेक्शन खोलता है, और हर बंद कनेक्शन अपना स्थानीय पोर्ट `TIME_WAIT` में रोके रखता है; लोड में loopback पर प्रॉक्सी के ephemeral पोर्ट जल्दी समाप्त हो जाते हैं, और उसका थ्रूपुट गिर जाता है। एक लॉक के अधीन पूल हर प्रॉक्सी अनुरोध को उस लॉक के लिए होड़ में डाल देता। इसलिए पूल (`crates/net/src/pool.rs`) को पट्टियों (stripes) में बाँटा गया है: 16 सॉकेट की 16 पट्टियाँ, हर पट्टी अपनी अलग cache line पर एक `Mutex<Vec<Arc<TcpStream>>>`। पहले उपयोग पर हर थ्रेड को round robin से एक मूल पट्टी (home stripe) मिलती है, वह पहले उसी में देता और उसी से लेता है, और नया कनेक्शन बनाने से पहले, किसी पकड़े हुए लॉक की प्रतीक्षा किए बिना, दूसरी पट्टियों में देखता है।

```
pub fn take(&self) -> Option<Arc<TcpStream>> {
    if self.is_stopping() {
        return None;
    }
    let home = home_stripe();
    if let Some(socket) =
        self.stripe(home).and_then(|stripe| lock(&stripe.sockets).pop())
    {
        return Some(socket);
    }
    (1..STRIPES).find_map(|offset| {
        let stripe = self.stripe((home + offset) % STRIPES)?;
        try_lock(&stripe.sockets)?.pop()
    })
}
```

पूल का कोई सॉकेट निष्क्रिय रहते समय ऑरिजिन द्वारा बंद किया जा चुका हो सकता है। प्रॉक्सी इसे दो में से एक तरह से पहचानता है, विफल राइट या पहले रिस्पॉन्स बाइट से पहले end-of-stream, और दोनों स्थितियों में क्लाइंट तक अभी कुछ नहीं पहुँचा होता, इसलिए वह सॉकेट बंद करता है और एक नए कनेक्शन पर एक बार रीट्राई करता है, कभी पूल के किसी दूसरे सॉकेट पर नहीं: निष्क्रिय अवधि के बाद पूल का हर सॉकेट पुराना पड़ चुका हो सकता है, और इस रीट्राई का पहला संस्करण, जो पूल का दूसरा सॉकेट लेता था, हर शांत अवधि के बाद पहले अनुरोध को `502` उत्तर देता था। रीट्राई केवल सुरक्षित मेथड (`GET`, `HEAD`, `OPTIONS`) के लिए अनुमत है जिसकी बॉडी पूरी तरह बफ़र हुई हो। उत्तर न मिलना इस बात का साक्ष्य नहीं कि ऑरिजिन ने कोई परिवर्तन कमिट नहीं किया, और टुकड़ों में रिले की गई बॉडी दोबारा नहीं भेजी जा सकती; दोनों स्थितियों में क्लाइंट को `502` मिलता है।

```
let safe = matches!(request.method, Method::Get | Method::Head | Method::Options);
let buffered = exchange.upload.is_buffered(request.body.len());
let retryable = safe && buffered;
for attempt in 0..2 {
    // रीट्राई हमेशा नया कनेक्शन बनाता है: निष्क्रिय अवधि के बाद पूल का हर
    // सॉकेट पुराना पड़ चुका हो सकता है, और दूसरा पुराना सॉकेट अनुरोध विफल कर देगा।
```

```
    let pooled = if attempt == 0 { pool.take() } else { None };
    // ... पूल में कुछ न हो तो कनेक्ट करें, activity से जोड़ें, एक्सचेंज करें
    match outcome {
        Ok(relayed) => {
            if relayed.origin_reusable {
                pool.give(origin);
            }
            return Ok(relayed.client_keep && buffered);
        }
        Err(error) => {
            drop(origin);
            let stale = was_pooled
                && retryable
                && matches!(
                    error,
                    ProxyError::UpstreamClosed | ProxyError::UpstreamWriteFailed
                );
            // ... बाकी हर त्रुटि तुरंत रिपोर्ट होती है
        }
    }
}
```

यूनिट परीक्षण निश्चित बाइट स्ट्रिंग को इन्हीं फ़ंक्शन से रिले करते हैं और पुष्टि करते हैं कि कनेक्शन हेडर को छोड़कर आउटपुट byte-identical है (`crates/net/src/relay/tests.rs`)। डेमन का एक end-to-end परीक्षण एक क्लाइंट सॉकेट पर दो प्रॉक्सी अनुरोध भेजता है और स्टब ऑरिजिन द्वारा जोड़े गए एक क्रम हेडर से जाँचता है कि दूसरे ने पूल का ऑरिजिन कनेक्शन दोबारा उपयोग किया। `crates/net/tests/` में एक vector परीक्षण दर्ज किए गए प्रॉक्सी परिदृश्यों (एक कच्चा क्लाइंट अनुरोध और एक पहले से तैयार ऑरिजिन रिस्पॉन्स) को दोबारा चलाता है और ऑरिजिन तथा क्लाइंट को मिलने वाले हर बाइट की तुलना करता है।

> **ऑरिजिन सॉकेट एक `Arc` क्यों है** सॉकेट डिस्क्रिप्टर एक छोटा पूर्णांक है जिसे कर्नेल बंद होते ही दोबारा उपयोग कर लेता है। शटडाउन को चल रहे एक्सचेंज को बाधित करना होता है, फिर भी उसे कभी ऐसे डिस्क्रिप्टर को नहीं छूना चाहिए जो इस बीच किसी दूसरे कनेक्शन के लिए दोबारा उपयोग हो चुका हो। लॉक के अधीन सक्रिय डिस्क्रिप्टर की एक टेबल इससे बचा सकती थी, हर एक्सचेंज पर एक लॉक की कीमत पर। Sibuna को ऐसी किसी टेबल की ज़रूरत नहीं: ऑरिजिन सॉकेट एक `Arc<TcpStream>` है। पूल एक हैंडल रखता है, एक्सचेंज एक रखता है, और एक्सचेंज चलते समय कनेक्शन का `Activity` एक clone रखता है। डिस्क्रिप्टर अंतिम हैंडल छोड़े जाने पर बंद होता है, इसलिए जब तक कोई उस तक पहुँच सकता है, वह न बंद हो सकता है, न दोबारा उपयोग हो सकता है।

निष्क्रियता की समय-सीमाएँ

हर क्लाइंट कनेक्शन अपना सॉकेट एक `IdleTable` (`crates/net/src/idle.rs`) में पंजीकृत करता है और एक `Lease` पाता है। lease का `Activity` वह समय रखता है जब कनेक्शन पर पिछली बार बाइट चले। डेमन हर सर्व किए गए अनुरोध के बाद इसे छूता है; प्रॉक्सी हर ऑरिजिन रीड पर और अपलोड के हर 16 KiB पर इसे छूता है, और चल रहे एक्सचेंज का ऑरिजिन सॉकेट इससे जोड़ता है। रीपर थ्रेड सबसे छोटे टाइमआउट के हर चौथाई पर (200 ms और एक सेकंड के बीच) जागता है और अपने टाइमआउट से अधिक समय तक चुप रहे हर कनेक्शन को, जुड़े हुए ऑरिजिन सॉकेट के साथ, बंद कर देता है, ताकि किसी भी ओर अवरुद्ध थ्रेड end of stream देखे और अपना स्लॉट छोड़ दे।

इसलिए समय-सीमाएँ चुप्पी को मापती हैं, रिस्पॉन्स की लंबाई को कभी नहीं: चलता रहने वाला बड़ा डाउनलोड कभी नहीं कटता। अपलोड के बाइट अविश्वसनीय पक्ष से आते हैं और केवल हर 16 KiB पर stamp को ताज़ा करते हैं, जो क्लाइंट से हर निष्क्रिय अवधि में कम से कम 16 KiB माँगता है; यह न्यूनतम-दर वाले अनुरोध टाइमआउट की slow-body रक्षा है।

टेबल में प्रति स्लॉट एक छोटा लॉक है और पूरी टेबल पर कोई लॉक नहीं। यह हर क्लाइंट सॉकेट का एक `try_clone` और हर जुड़े ऑरिजिन सॉकेट का एक `Arc` रखती है, ऊपर बताए गए कारण से। `Lease` अपने कनेक्शन को `Drop` में अपंजीकृत करता है, और रीप किया गया स्लॉट तब तक अपने lease का ही रहता है, इसलिए कोई पुराना कनेक्शन कभी वह स्लॉट नहीं छोड़ सकता जिसे किसी नए ने ले लिया हो।

अपग्रेड किए गए कनेक्शन

सत्यापित `101` के बाद WebSocket अपग्रेड दोनों दिशाओं में रिले होता है। हर दिशा एक निश्चित बफ़र के साथ अपने अलग थ्रेड पर चलती है: कनेक्शन का अपना थ्रेड ऑरिजिन से क्लाइंट की ओर के बाइट ले जाता है, और 128 KiB स्टैक वाला एक scoped थ्रेड क्लाइंट से ऑरिजिन की ओर के बाइट ले जाता है। एक थ्रेड `poll(2)` से दोनों दिशाओं की सेवा कर सकता था, लेकिन Rust की standard library इसे उपलब्ध नहीं कराती; सॉकेट टाइमआउट वाले दो थ्रेड रिले को `unsafe` से मुक्त रखते हैं। इसलिए अपग्रेड किया गया कनेक्शन दो थ्रेड रखता है, और जब हर कनेक्शन अपग्रेड हो तो कनेक्शन थ्रेड की संख्या `--max-connections` की दोगुनी तक पहुँच सकती है।

दोनों दिशाएँ एक निष्क्रियता समय-सीमा साझा करती हैं (`--websocket-idle-timeout`, डिफ़ॉल्ट 300 सेकंड): किसी भी ओर चलने वाले बाइट एक stamp को ताज़ा करते हैं। हर सॉकेट प्रतीक्षा बचे हुए समय के बराबर टाइमआउट रखती है; टाइमआउट होने वाली प्रतीक्षा साझा stamp को दोबारा पढ़ती है और या तो शेष समय के लिए फिर प्रतीक्षा करती है या रिले समाप्त कर देती है। जब एक दिशा end of stream पर पहुँचती है, तो दूसरे सॉकेट की भेजने वाली ओर बंद कर दी जाती है (half close); जब कोई दिशा विफल होती है, तो दोनों सॉकेट बंद कर दिए जाते हैं ताकि दूसरी दिशा भी समाप्त हो। अपग्रेड प्रॉक्सी होने से पहले कनेक्शन अपना cache किया गया नीति इंजन छोड़ देता है, ताकि घंटों तक चल सकने वाला कनेक्शन किसी पुराने इंजन को जीवित न रखे।

रोकना

`Running::stop`, और इसी तरह `Running` का `Drop`, डेमन को एक निश्चित क्रम में रोकता है। यह stopping फ़्लैग सेट करता है और listener से जुड़कर हर accept वर्कर को जगाता है, वर्करों को join करता है, हर पंजीकृत क्लाइंट सॉकेट और उससे जुड़े ऑरिजिन सॉकेट को बंद करता है (`IdleTable::shutdown`), ऑरिजिन पूल को रोकता है, हर कनेक्शन थ्रेड को join करता है, रीपर को join करता है, और पूल के सॉकेट बंद करता है।

SIGINT और SIGTERM साधारण Rust के ज़रिए इस क्रम तक पहुँचते हैं। कोई भी थ्रेड शुरू करने से पहले डेमन दोनों सिग्नल को ब्लॉक कर देता है (बाद का हर थ्रेड यह mask विरासत में पाता है) और एक थ्रेड को `sigwait` के लिए समर्पित करता है, जो अनुरोध दर्ज करता है और मुख्य थ्रेड को तुरंत जगाता है। किसी signal handler में कोई कोड नहीं चलता, और कोई थ्रेड किसी फ़्लैग को बार-बार नहीं जाँचता। signal मॉड्यूल (`cmd/sibuna/src/signals.rs`) में डेमन का एकमात्र `unsafe` कोड है: उन कुछ C library फ़ंक्शन की घोषणाएँ जिन्हें `std` रैप नहीं करता।

> **`Drop` द्वारा मुक्ति** स्पष्ट cleanup कॉल सेटअप के उलटे क्रम में लिखनी पड़ती हैं, और हर जल्दी return को उनके विरुद्ध पढ़ना पड़ता है। Sibuna में हर संसाधन `Drop` में स्वयं को मुक्त करता है: `Daemon` और `Running` सर्वर रोकते हैं, `Lease` अपने कनेक्शन को अपंजीकृत करता है, `Counted` कनेक्शन सीमा में एक स्थान लौटाता है, और `Finished` एक task स्लॉट मुक्त करता है। कंपाइलर हर मुक्ति को scope से बाहर जाने वाले हर पाथ पर डालता है, उससे होकर गुज़रने वाले panic unwinding सहित, इसलिए कोई पाथ इसे भूल नहीं सकता।

हमने क्या मापा

तालिका एक Ryzen 9 5950X पर रिवर्स-प्रॉक्सी मोड में Shield को दिखाती है: डेमन चार CPU पर पिन किया गया, `wrk` दो थ्रेड और 64 कनेक्शन के साथ आठ अन्य CPU पर, पाँच-सेकंड के रन, तीन की माध्यिका। पंक्तियाँ `benchmarks/results/product-latest.md` से कॉपी की गई हैं; पूरा रिकॉर्ड भाग VIII में है।

तालिका 7 रिवर्स-प्रॉक्सी मोड में Shield, `benchmarks/results/product-latest.md` से

वर्कलोड	स्टेटस	req/s	p50 / p99 (us)	प्रति अनुरोध CPU (us)
सत्र के साथ स्वीकृत, प्रॉक्सी किया गया	200	138,965	438 / 857	28.57
चैलेंज किया गया (इंटरस्टिशियल)	200	195,188	162 / 328	13.73
स्टैटिक अनुमत, प्रॉक्सी किया गया	200	145,673	415 / 746	27.27
WAF द्वारा अस्वीकृत (सत्र के साथ)	403	209,507	151 / 306	12.17

`wrk` 64 अनुरोध एक साथ चलते रखता है, इसलिए CPU पर निर्भर प्रॉक्सी पाथ पर अनुरोध चार CPU के लिए कतार में लगते हैं, और वहाँ का p99 एक अनुरोध की लागत नहीं बल्कि शेड्यूलर की टेल है। इसके बजाय open-loop जनरेटर, जो एक निश्चित दर से भेजता है और हर अनुरोध का समय उसके इच्छित भेजने के क्षण से मापता है, समान प्रस्तावित लोड पर लेटेंसी मापता है। रिवर्स-प्रॉक्सी मोड में स्वीकृत अनुरोधों के लिए p99 प्रति सेकंड 10,000 अनुरोधों पर 65 us और 100,000 पर 100 us है; 150,000 प्रस्तावित पर चार CPU संतृप्त हो जाते हैं, प्रति सेकंड 134,344 अनुरोध प्राप्त होते हैं, और कतार सेकंडों तक बढ़ जाती है (`benchmarks/results/latency-latest.md`)। warm keep-alive कनेक्शन हर निर्णय हीप आवंटन के बिना सर्व करता है: `cmd/sibuna/tests/steady_state.rs` डेमन के आवंटन गिनता है, और `crates/net/tests/steady_state.rs` 500 प्रॉक्सी किए गए keep-alive अनुरोधों में कोई आवंटन नहीं पाता।

अभ्यास 5.1. एक क्लाइंट 200 KB का अपलोड भेजता है। ट्रेस करें कि कौन-से बाइट 64 KiB बफ़र में रहते हैं, कौन-से `relay_body` द्वारा अपलोड बफ़र से रिले होते हैं, और प्रॉक्सी को रिले करने से पहले हेड ऑरिजिन को क्यों लिखना होगा।

आवंटन के बिना समवर्ती अवस्था

उद्देश्य शार्ड लॉक, Robin Hood spent set, GCRA लिमिटर, संस्करण वाली बैन टेबल, MPSC घटना रिंग, और `Arc` तथा generation-checked cache के साथ नीति इंजन के read-copy-update प्रकाशन का विश्लेषण करें।

`sibuna-store` का हर स्टोर अपनी निश्चित क्षमता एक बार, बनते समय, आवंटित करता है, और उसके बाद न कभी आवंटन करता है और न किसी वैश्विक लॉक पर रुकता है। किसी को बैकग्राउंड sweeper की ज़रूरत नहीं: समाप्त हुई अवस्था उसी स्थान पर वापस ली जाती है।

लॉक और शार्डिंग

Sibuna की टेबल के critical section दसियों निर्देश लंबे हैं, कर्नेल futex के round trip से कहीं छोटे। टेबल को कुंजी हैश से 16 शार्ड में बाँटा गया है, इसलिए दो ऑपरेशन 1/16 प्रायिकता से होड़ करते हैं, और हर शार्ड अपनी अलग cache line पर है, इसलिए पड़ोसी लॉक false sharing नहीं करते।

टेबल `std::sync::Mutex` का उपयोग करती हैं: होड़ न होने पर एक compare-and-swap, धारक के चलते समय थोड़ा spin, फिर futex प्रतीक्षा। विकल्प एक दो-अवस्था वाला atomic spin lock है जिसके प्रतीक्षा लूप में spin-loop hint हो। SID 0010 ने हमसे मापन के आधार पर इन दोनों में से चुनने को कहा। spin lock होड़-रहित ऑपरेशन पर लगभग एक नैनोसेकंड बचाता है, लेकिन होड़ में हार जाता है:

तालिका 8 Ryzen 9 5950X पर शार्ड लॉक का थ्रूपुट (अक्टूबर 2026)

वर्कलोड	`Mutex`	Spin lock
spent set, 16 थ्रेड	10.8 M/s	8.6 M/s
spent set, 64 थ्रेड (CPU से अधिक)	11.2 M/s	6.9 M/s
छह व्यस्त पतों पर GCRA, 16 थ्रेड	44.9 M/s	19.3 M/s
छह व्यस्त पतों पर GCRA, 64 थ्रेड	37.4 M/s	6.5 M/s

edge सर्वर CPU से अधिक कनेक्शन थ्रेड चलाता है, और जिस spin lock का धारक preempt हो जाए वह हर प्रतीक्षक से उसका time slice जलवाता है, इसलिए हम mutex रखते हैं। poisoned लॉक में प्रवेश किया जाता है, उसे आगे नहीं बढ़ाया जाता: कोई ऑपरेशन शार्ड पकड़े हुए panic नहीं करता, और यदि करे भी तो शार्ड संरचनात्मक रूप से वैध रहता है।

Robin Hood spent set

केवल स्वीकृत प्रमाण spent set में आते हैं (भाग III)। इसके 16 शार्ड हैं, हर एक `Option<Entry>` के 4,096 स्लॉट की open-addressed ऐरे, जहाँ एक प्रविष्टि `{ tag: [u8; 16], expires_at: u64, dist: u8 }` है। Robin Hood insertion (Celis, 1986) हर उस निवासी को हटा देता है जो आने वाली कुंजी की तुलना में अपने घर (home) के अधिक निकट है, जिससे probe की लंबाइयाँ एक-दूसरे के पास बनी रहती हैं; लुकअप जैसे ही ऐसी प्रविष्टि से मिलता है जो अपने घर के उससे अधिक निकट है जितनी खोजी जा रही कुंजी अपने घर के, वह वहीं रुक जाता है।

> **इम्प्लीमेंटेशन: `Table::insert`** एक spent टैग को उसकी समाप्ति के साथ डालता है, रास्ते में समाप्त स्लॉट वापस लेते हुए। `crates/store/src/spent.rs` में (क्रेट `sibuna-store`)।

```rust
fn insert(
    &mut self,
    tag: &Tag,
    expires_at: u64,
    now: u64,
) -> Result<(), SpendError> {
    if !self.can_insert(tag, now) {
        return Err(SpendError::Full);
    }
    let mut carry = Entry { tag: *tag, expires_at, dist: 0 };
```

```
    let mut idx = Self::home(tag);
    while carry.dist < MAX_PROBE {
        let slot = &mut self.entries[idx];
        let Some(e) = slot else {
            *slot = Some(carry);
            return Ok(());
        };
        // समाप्त निवासी को अधिलेखित करना तभी सुरक्षित है जब नई दूरी
        // छोटी न हो, ताकि इस स्लॉट से आगे probe करने वाली कुंजियाँ
        // early-termination जाँच में फिर भी पास हों।
        if e.is_expired(now) && carry.dist >= e.dist {
            *e = carry;
            return Ok(());
        }
        if carry.dist > e.dist {
            std::mem::swap(&mut carry, e);
            // हटाई गई समाप्त प्रविष्टि बस छोड़ दी जाती है।
            if carry.is_expired(now) {
                return Ok(());
            }
        }
        idx = (idx + 1) % SHARD_CAPACITY;
        carry.dist += 1;
    }
    Err(SpendError::Full)
}
```

समाप्त-स्लॉट वाला नियम सूक्ष्म भाग है: समाप्त प्रविष्टि को उसी स्थान पर केवल ऐसी कुंजी अधिलेखित कर सकती है जो अपने घर से कम से कम उतनी ही दूर हो, अन्यथा उस स्लॉट से आगे probe करने वाली कोई बाद की कुंजी जल्दी रुक जाएगी और खो जाएगी। हटाई गई समाप्त प्रविष्टियाँ बस छोड़ दी जाती हैं। किसी sweeper थ्रेड की ज़रूरत नहीं। प्रविष्टि पहले कुछ भी बदले बिना विस्थापन पाथ का अनुकरण करती है (`can_insert`) और जाँचती है कि सीमित विस्थापन किसी जीवित टैग को खोए बिना सफल हो सकता है।

> **अपरिवर्तनीय नियम** अस्वीकृत प्रविष्टि कभी किसी जीवित टैग को नहीं हटाती। इसलिए जो टैग खर्च हो चुका है और समाप्त नहीं हुआ, वह हमेशा मिल जाता है, और अस्वीकृति (`SpendError::Full`) सेवा से इनकार तो कर सकती है लेकिन कभी रीप्ले संभव नहीं बनाती।

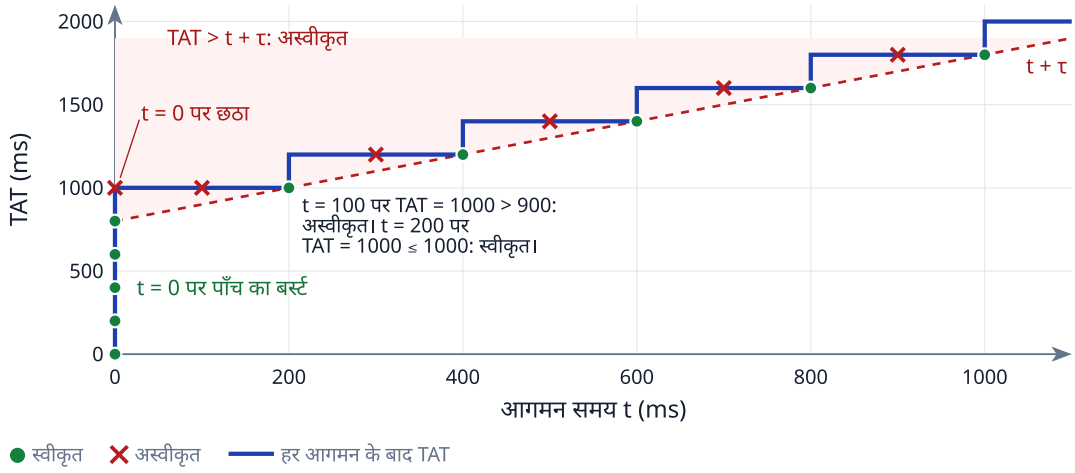
एक स्लॉट `Option<Entry>` है। खाली स्लॉट में पढ़ने के लिए कोई टैग नहीं होता, इसलिए कोई `occupied` फ़्लैग नहीं है जिसे जाँचना कोई पाठक भूल सके, और `match` तब तक कंपाइल नहीं होगा जब तक दोनों स्थितियाँ संभाली न जाएँ। त्रुटि का प्रकार भी उतना ही सटीक है: spent set दो तरह से विफल हो सकता है (`DoubleSpend` और `Full`), और `SpendError` ठीक इन्हीं के नाम देता है।

GCRA: एक पूर्णांक में रेट लिमिटिंग

Generic Cell Rate Algorithm (ATM Forum, 1996) leaky bucket का virtual-scheduling रूप है। प्रति क्लाइंट यह एक सैद्धांतिक आगमन समय (TAT) संग्रहीत करता है। प्रति विंडो W धनात्मक सीमा N के लिए उत्सर्जन अंतराल $T = \max(1, \lceil W/N \rceil)$ और बर्स्ट टॉलरेंस $\tau = (N-1)T$ के साथ:

$$\text{arrival at } t \text{ conforms} \iff \mathrm{TAT} \le t + \tau, \quad \text{then } \mathrm{TAT} \leftarrow \max(\mathrm{TAT}, t) + T.$$

प्रमेय (token-bucket सीमा) t_1 पर पहले अनुरूप (conforming) आगमन के बाद k-वाँ अनुरूप आगमन $t_k - t_1 \geq (k-1)T - \tau$ को संतुष्ट करता है। इसलिए L लंबाई के किसी भी अंतराल में अधिकतम $N + \lfloor L/T \rfloor$ अनुरूप आगमन होते हैं: N का एक बर्स्ट, फिर प्रति T एक। परिभाषित बर्स्ट से आगे निश्चित विंडो की सीमा का कोई दोष नहीं होता, और प्रति-अनुरोध इतिहास नहीं रखना पड़ता।



चित्र 7 प्रति सेकंड 5 की सीमा के लिए GCRA प्रवेश: पाँच का बर्स्ट, फिर हर 200 ms पर एक। कोई आगमन तब स्वीकृत होता है जब उसे मिलने वाला TAT रेखा $t + \tau$ पर या उसके नीचे हो।

हल किया गया उदाहरण: हाथ से प्रति सेकंड पाँच $N = 5$ और $W = 1000$ ms लें, ताकि $T = 200$ ms और $\tau = 800$ ms हों, और मान लें कि एक निष्क्रिय रहा क्लाइंट $t = 0$ पर छह अनुरोध भेजता है। पहले को खाली सेल मिलती है: $\mathrm{TAT} = \max(0, 0) = 0 \leq 800$, इसलिए वह अनुरूप है और $\mathrm{TAT} \leftarrow 200$। अगले चार इसी तरह अनुरूप हैं और $\mathrm{TAT} = 1000$ छोड़ते हैं। छठे को $1000 > 0 + 800$ मिलता है और उसे `Retry-After` $\mathrm{TAT} - \tau - t = 200$ ms के साथ अस्वीकार किया जाता है। $t = 199$ पर अनुरोध को अब भी $1000 > 999$ मिलता है और वह अस्वीकार होता है; $t = 200$ पर उसे $1000 \leq 1000$ मिलता है, वह अनुरूप होता है, और $\mathrm{TAT} = 1200$ छोड़ता है। `crates/store/src/limiter.rs` में यूनिट परीक्षण `gcra_admits_exactly_the_burst_then_drains_at_the_emission_rate` ठीक इन्हीं संख्याओं की जाँच करता है।

इम्प्लीमेंटेशन: `arrive` लॉक किए गए शार्ड की सेल पर एक आगमन लागू करता है और निर्णय, रीट्राई की देरी और शेष बजट लौटाता है। `crates/store/src/limiter.rs` में (क्रेट `sibuna-store`)।

```
fn arrive(
    cells: &mut [Cell; SLOTS_PER_SHARD],
    key: u64,
    now_ms: u64,
    limits: Limits,
```

```
) -> Decision {
    let interval = limits.emission_interval();
    let tau = limits.burst_tolerance();
    let Some(cell) = locate(cells, key, now_ms, tau) else {
        return Decision {
            limited: true,
            capacity_exhausted: true,
            retry_after_ms: interval,
            remaining: 0,
        };
    };
    let tat = cell.tat_ms.max(now_ms);
    let horizon = now_ms.saturating_add(tau);
    if tat > horizon {
        return Decision {
            limited: true,
            capacity_exhausted: false,
            retry_after_ms: tat - tau - now_ms,
            remaining: 0,
        };
    }
    cell.tat_ms = tat.saturating_add(interval);
    // ... remaining = min(rate, floor((t + tau - TAT) / T) + 1)
}
```

शार्ड का `check` लॉक लेता है और लॉक की गई सेल के साथ `arrive` को कॉल करता है; `check` लौटने पर guard लॉक छोड़ देता है। सेल 16 बाइट की हैं (एक कुंजी और एक TAT), 512 स्लॉट के 16 शार्ड में, 16 स्लॉट की probe विंडो के साथ, कुल लगभग 128 KiB। जिस सेल का TAT $t - \tau$ से पुराना है, वह खाली हो चुकी है और तुरंत वापस ली जाती है। `Retry-After` हेडर उसी अंकगणित से निकाला जाता है। भरी हुई probe विंडो सक्रिय कोटा अवस्था को हटाने के बजाय नए क्लाइंट को अस्वीकार करती हैं, और निर्णय यह बताता है (`capacity_exhausted`), ताकि कॉल करने वाला क्षमता-जनित अस्वीकृति को प्रतिबंध में न बदले। शून्य पर कॉन्फ़िगर की गई दर अनुरोधों को अस्वीकार करती है। एक लिमिटर कई नियमों की सेवा करता है: सेल कुंजी नियम के scope से seed किया गया क्लाइंट पते का Wyhash है, इसलिए अलग-अलग scope में एक ही क्लाइंट के स्वतंत्र बजट होते हैं।

संस्करण वाली बैन टेबल

प्रतिबंध atomic संस्करण, कुंजी और समाप्ति फ़ील्ड वाले 4,096 स्लॉट उपयोग करते हैं, जो प्रतिबंधित पते के कुंजी-युक्त हैश से 8 स्लॉट की probe विंडो के साथ open-addressed हैं। एक writer, जो एक mutex द्वारा दूसरे writers के साथ क्रम में लगाया जाता है, हर अपडेट को संस्करण वृद्धियों के बीच रखता है; पाठक कोई लॉक नहीं लेते, केवल एक स्थिर सम (even) संस्करण स्वीकार करते हैं, और समवर्ती बदलावों पर दोबारा प्रयास करते हैं।

> **इम्प्लीमेंटेशन: `BanTable::ban` और `BanTable::read_slot`** writer का कोष्ठक (bracket) और पाठक का सुसंगत स्नैपशॉट। `crates/store/src/ban_table.rs` में (क्रेट `sibuna-store`)।

```
pub fn ban(&self, banned: &[u8], until: u64, now: u64) {
    let key = slot_key(banned);
    let _writers = self.lock_writers();
    let victim = &self.slots[self.victim(key, now)];
    // कुंजी और समाप्ति एक तार्किक मान हैं: बदलाव को विषम संस्करण के बीच
```

```
    // रखें, ताकि पाठक पुरानी कुंजी को अगली कुंजी की समाप्ति के साथ
    // न जोड़ सकें।
    victim.version.fetch_add(1, Ordering::SeqCst);
    victim.until.store(until, Ordering::SeqCst);
    victim.key.store(key, Ordering::SeqCst);
    victim.version.fetch_add(1, Ordering::SeqCst);
}

fn read_slot(&self, slot: &Slot) -> (u64, u64) {
    loop {
        let before = slot.version.load(Ordering::SeqCst);
        if before & 1 != 0 {
            std::hint::spin_loop();
            continue;
        }
        let key = slot.key.load(Ordering::SeqCst);
        let until = slot.until.load(Ordering::SeqCst);
        if before == slot.version.load(Ordering::SeqCst) {
            return (key, until);
        }
    }
}
```

अनुक्रमिक सुसंगतता (sequential consistency) पुरानी पहचान को नई समाप्ति के साथ मिलने से रोकती है: हर पहुँच एक कुल क्रम (total order) का हिस्सा है, जो तर्क को सरल रखता है। पाठक कोई mutex नहीं लेते, लेकिन किसी writer की प्रतीक्षा कर सकते हैं और wait-free नहीं हैं। इन atomic ऑपरेशन की वास्तविक लागत है, जिसे मुफ़्त मानने के बजाय HTTP हार्नेस में अप्रत्यक्ष रूप से मापा जाता है। भरी हुई probe विंडो किसी समाप्त हो चुके स्लॉट को, अन्यथा सबसे पहले समाप्त होने वाले स्लॉट को, अधिलेखित करती है। `clear` हर प्रतिबंध हटाता है लेकिन कुंजियों को उनकी जगह छोड़ देता है, क्योंकि खाली किया गया स्लॉट probe chain में आगे की किसी टकराने वाली कुंजी को छिपा सकता है।

MPSC घटना रिंग

जब Shield सतह किसी अनुरोध को अस्वीकार करती है या हनीपॉट सक्रिय होता है, तो घटना को लगभग 1.3 KB के एक निश्चित रिकॉर्ड में कॉपी किया जाता है (क्लाइंट पता, User-Agent, मेथड, पाथ, श्रेणी और पेलोड, हर एक अपनी क्षमता पर काटा गया) और एक सीमित multi-producer single-consumer रिंग पर push किया जाता है (Vyukov का sequence-stamped डिज़ाइन, 512 स्लॉट, `BoundedQueue<IncidentRecord, 512>`)। producer कनेक्शन थ्रेड हैं; consumer स्टोरेज थ्रेड है। भरी हुई रिंग रिस्पॉन्स को रोकने के बजाय सबसे नया रिकॉर्ड छोड़ देती है, यह टिकाऊपन की गारंटी नहीं बल्कि एक स्पष्ट हानि-नीति है। drop काउंटर उस हानि को दृश्य बनाता है।

```
/// `incident` को रिंग में कॉपी करता है; भरी हुई रिंग उसे छोड़ देती है और drop गिनती है।
pub(crate) fn record(&self, incident: &Incident<'_>) {
    if self.queue.push(IncidentRecord::from_incident(incident)).is_err() {
        self.metrics.incidents_dropped.bump();
    }
}
```

किसी रिकॉर्ड को लंबित बैच में ले जाने के बाद स्टोरेज थ्रेड उसे, तैयार पंक्तियों के साथ, कमिट की पुष्टि होने तक रखता है; वह डेटाबेस त्रुटि पर उसे चुपचाप नहीं फेंकता, और रीट्राई वही बैच लिखता है।

हर स्लॉट एक क्रम संख्या रखता है जो producers को बताती है कि स्लॉट मुक्त है या नहीं और consumer को बताती है कि वह भरा है या नहीं, इसलिए कोई भी पक्ष लॉक नहीं लेता। ऐसी रिंग safe Rust में नहीं लिखी जा सकती: producers और consumer एक ही ऐरे के अलग-अलग स्लॉट को एक ही समय छूते हैं, और कौन-सा थ्रेड कौन-सा स्लॉट छू सकता है,

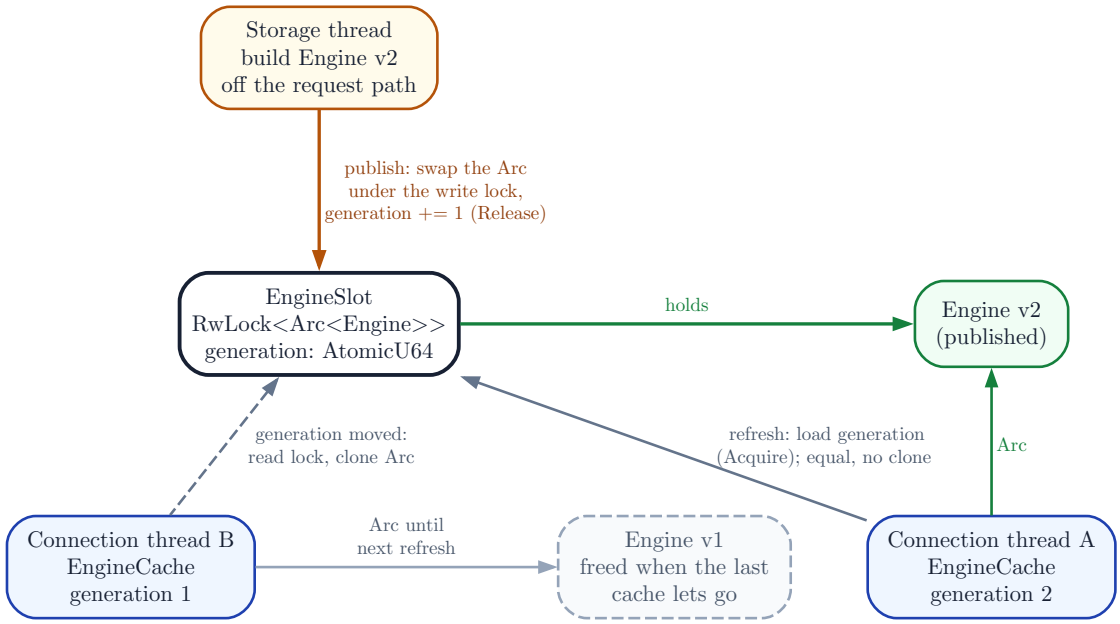
यह type system नहीं, केवल क्रम प्रोटोकॉल सिद्ध करता है। इसलिए मॉड्यूल `crates/store/src/ring.rs` अपने मान `UnsafeCell<MaybeUninit<T>>` स्लॉट में रखता है, और हर `unsafe` ब्लॉक प्रोटोकॉल के उस चरण का नाम देता है जो उसे उचित ठहराता है:

```
if diff == 0 {
    // Relaxed: CAS केवल यह तय करता है कि स्थिति `pos` किस producer की है।
    match self.tail.0.compare_exchange_weak(
        pos, pos.wrapping_add(1), Ordering::Relaxed, Ordering::Relaxed,
    ) {
        Ok(_) => {
            // SAFETY: CAS ने इस थ्रेड को स्थिति `pos` दी है, और
            // `sequence == pos` बताता है कि स्लॉट खाली है; कोई दूसरा producer
            // `pos` का दावा नहीं कर सकता, और consumer नीचे के Release store से
            // पहले उसे नहीं पढ़ता।
            unsafe { (*slot.value.get()).write(value) };
            // Release: लिखा गया मान consumer के लिए प्रकाशित करता है।
            slot.sequence.store(pos.wrapping_add(1), Ordering::Release);
            return Ok(());
        }
        Err(current) => pos = current,
    }
} else if diff < 0 {
    return Err(value);
}
```

रिंग की दो परिपाटियाँ प्रकार (type) हैं, टिप्पणी नहीं। भरी हुई रिंग पर push `Err(value)` लौटाता है, रिकॉर्ड वापस सौंपते हुए ताकि कॉल करने वाला तय करे कि उसे गिने या छोड़े; एक कोरा `false` उसे खो देता। रिंग का एक ही consumer है, और pop `&mut self` लेता है, और साझा कोड को पहले अद्वितीय `Consumer` पर दावा करना होता है, इसलिए दूसरा consumer या तो कंपाइल नहीं होता या रन टाइम पर अस्वीकार होता है।

इंजन का read-copy-update प्रकाशन

नीतियाँ बदलने पर नीति इंजन हॉट पाथ से बाहर दोबारा बनाया जाता है। अनुरोधों को कभी आधा बना इंजन नहीं दिखना चाहिए, और कोई इंजन तब कभी मुक्त नहीं होना चाहिए जब कोई अनुरोध उसे अभी पढ़ रहा हो। Sibuna हर इंजन को एक `EngineSlot` में `Arc<Engine>` के रूप में प्रकाशित करता है, और हर कनेक्शन उसे एक `EngineCache` के ज़रिए पढ़ता है:



चित्र 8 इंजन प्रकाशन। स्टोरेज थ्रेड एक पूरा इंजन प्रकाशित करता है और generation आगे बढ़ाता है; हर कनेक्शन एक acquire load से generation जाँचता है और प्रकाशित Arc को केवल किसी प्रकाशन के बाद clone करता है। पुराना इंजन तब मुक्त होता है जब उसे रखने वाला अंतिम cache उसे छोड़ देता है।

इम्प्लीमेंटेशन: `EngineSlot::publish` और `EngineCache::refresh` पूरे इंजन का प्रकाशन, और एक कनेक्शन की ओर से उसका दृश्य। `cmd/sibuna/src/slot.rs` में (क्रेट `sibuna`)।

```
pub struct EngineSlot {
    current: RwLock<Arc<Engine>>,
    generation: AtomicU64,
}

pub fn publish(&self, engine: Engine) {
    let fresh = Arc::new(engine);
    let previous = {
        let mut current =
            self.current.write().unwrap_or_else(PoisonError::into_inner);
        std::mem::replace(&mut *current, fresh)
    };
    self.generation.fetch_add(1, Ordering::Release);
    // पुराना इंजन लॉक के बाहर छोड़ा जाता है (जब यह उसका अंतिम धारक था)।
    drop(previous);
}

pub struct EngineCache {
    generation: u64,
    engine: Option<Arc<Engine>>,
}
```

```rust
pub fn refresh(&mut self, slot: &EngineSlot) -> &Engine {
    let generation = slot.generation();
    if self.generation != generation {
        // ऊपर के load और इस clone के बीच हुआ प्रकाशन नए इंजन पर पुरानी
        // generation लगा देता है; अगला refresh बस दोबारा लोड करता है।
        self.generation = generation;
        self.engine = None;
    }
    self.engine.get_or_insert_with(|| slot.load())
}
```

इंजन अपनी जगह रखे जाने के बाद generation को release ordering से संग्रहीत किया जाता है, और acquire ordering से लोड किया जाता है: जो पाठक नई generation देखता है, वह उससे पहले संग्रहीत इंजन भी देखता है। टिप्पणी में बताई गई race हानिरहित है। यदि कोई प्रकाशन किसी पाठक के generation load और उसके clone के बीच पड़ता है, तो पाठक नए इंजन को पुराने generation नंबर के तहत cache कर लेता है, और उसका अगला refresh असंगति पकड़कर दोबारा लोड करता है। एक पाठक एक अनुरोध के लिए एक इंजन पीछे हो सकता है; वह कभी आधा बना इंजन नहीं देख सकता, क्योंकि स्लॉट में केवल पूरे इंजन ही संग्रहीत होते हैं।

अपरिवर्तनीय नियम हर इंजन जिसे कोई अनुरोध पढ़ता है, पूरा होता है, और कोई इंजन तब तक मुक्त नहीं होता जब तक कोई अनुरोध या कनेक्शन cache उसे रखे हुए है। पहली बात इसलिए सही है क्योंकि `publish` द्वारा संग्रहीत किए जाने से पहले इंजन पूरी तरह बन चुका होता है; दूसरी इसलिए क्योंकि हर पाठक एक `Arc` रखता है, और इंजन केवल अपने अंतिम `Arc` के साथ ही छोड़ा जाता है।

cache क्यों, केवल लॉक क्यों नहीं? SID 0010 ने हमसे 16 थ्रेड के साथ पढ़ने की लागत मापने और `RwLock<Arc<Engine>>` तथा safe Rust में एक lock-free विकल्प के बीच चुनने को कहा। लॉक के ज़रिए पढ़ने में lock word और reference count लिए जाते हैं, दो cache line जिन्हें हर कोर लिखता है। इसके विपरीत generation प्रति प्रकाशन एक बार लिखी जाती है और अन्यथा केवल पढ़ी जाती है, इसलिए हर कोर अपनी प्रति रखता है:

तालिका 9 एक इंजन रीड की लागत, Ryzen 9 5950X, लूप में पढ़ते 16 थ्रेड (अक्टूबर 2026)

पाठक	1 थ्रेड	16 थ्रेड
केवल `RwLock` read guard	4.2 ns	752 ns
`RwLock` read और `Arc` clone	7.6 ns	833 ns
cache किए गए `Arc` के साथ generation जाँच	0.8 ns	1.8 ns

होड़ में cache किया गया रीड लॉक से 400 गुना से अधिक सस्ता है, इसलिए अनुरोध पाथ cache का उपयोग करता है और `RwLock` का केवल refresh के धीमे पाथ पर। मापन `cmd/sibuna/src/slot.rs` में अनदेखा (ignored) परीक्षण `read_cost_under_contention` है। अनुरोध पाथ पर हर उपयोग refresh करता है, इसलिए अनुरोध उस चरण पर वर्तमान इंजन देखता है। इंजन स्वयं छोटा है: सघन हीप टेबल के 209,052 बाइट, लगभग 204 KiB (`benchmarks/results/primitives-latest.json` में `engine_bytes`), जिसमें 128 नियम, 128-बाइट नियम नाम और 8,192 trie नोड की क्षमता है।

`Arc` क्यों, reader count क्यों नहीं निश्चित-मेमोरी वाला विकल्प दो इंजन स्लॉट रखता है, हर एक का एक reader count और अपना आवंटन arena। एक अनुरोध उसकी गिनती बढ़ाकर और स्लॉट पॉइंटर को दोबारा जाँचकर स्लॉट को पिन करता है, और उस race को बंद करने के लिए दोनों atomic अनुक्रमिक रूप से सुसंगत होने चाहिए जिसमें कोई writer पाठक के load और उसकी वृद्धि के बीच स्लॉट बदल दे। builder पॉइंटर बदलता है और फिर तब तक spin करता है जब तक पुराने स्लॉट के पाठक खाली न हो जाएँ, उसके बाद ही वह उस स्लॉट का arena reset करके उसमें दोबारा बनाता है। इसलिए दोबारा बनने के दौरान स्लॉट पकड़े रखने वाला पाठक builder को रोक देता है, और धीमा ऑरिजिन अगले प्रकाशन को अटका देगा, जब तक कि हर अनुरोध प्रॉक्सी I/O शुरू होने से पहले अपना स्लॉट न छोड़ दे।

`Arc` की reference counting को इनमें से कुछ नहीं चाहिए। स्टोरेज थ्रेड हर स्नैपशॉट से एक नया `Engine` बनाता है और उसे प्रकाशित करता है; कुछ भी दोबारा उपयोग नहीं होता, इसलिए न कोई अतिरिक्त स्लॉट है, न arena reset, न spin: `publish` कभी पाठकों की प्रतीक्षा नहीं करता। पुराना इंजन बस तब तक जीवित रहता है जब तक उसका अंतिम धारक उसे नहीं छोड़ता, और धीमा ऑरिजिन केवल उस क्षण को टाल सकता है, अगले प्रकाशन को कभी नहीं। borrow समाप्त करने के लिए निर्णय मिलान हुए नियम का नाम एक 128-बाइट बफ़र में कॉपी करता है: निर्णय कनेक्शन के cache के ज़रिए इंजन को उधार लेता है, और रिस्पॉन्स पाथ को पूरा कनेक्शन संदर्भ mutable रूप में चाहिए। अपग्रेड किया गया कनेक्शन, जो घंटों चल सकता है, प्रॉक्सी होने से पहले अपना cache किया गया इंजन छोड़ देता है।

अभ्यास 5.2. GCRA प्रमेय का उपयोग करके `--rate-limit 100 --rate-window 10` के लिए निकालें कि निष्क्रिय रहने के बाद पहले 3 सेकंड में एक क्लाइंट अधिकतम कितने अनुरोध सफलतापूर्वक भेज सकता है। इसकी तुलना एक निश्चित 10-सेकंड विंडो काउंटर से और एक सच्चे sliding-window log से करें।

अपरिवर्तनीय नियम समझाएँ. तीन स्लॉट वाले उदाहरण से समझाएँ कि समाप्त Robin Hood प्रविष्टि को केवल कम से कम उतनी ही बड़ी दूरी वाली कुंजी ही क्यों अधिलेखित कर सकती है।

भाग VI

ऑटोमेटा, trie और सिमैंटिक फ़ायरवॉल

हम सब-माइक्रोसेकंड वर्गीकरण के पीछे के एल्गोरिदम की जाँच करते हैं: टैग किया गया *Aho-Corasick* ऑटोमेटन, *IPv4/IPv6 radix trie*, सिमैंटिक *WAF* के *byte-class* टोकनाइज़र, और *WEIGH* स्कोरिंग वाला घोषणात्मक नीति इंजन।

एक पास में अनेक पैटर्न का मिलान

उद्देश्य इस अध्याय के अंत तक हमें यह कर पाना चाहिए: Aho-Corasick ऑटोमेटन की रैखिक-समय गारंटी निकालना, सघन टेबल का लेआउट और byte class के ज़रिए case folding समझाना, और तर्क देना कि SIMD लिटरल इंजन क्यों नहीं अपनाया गया।

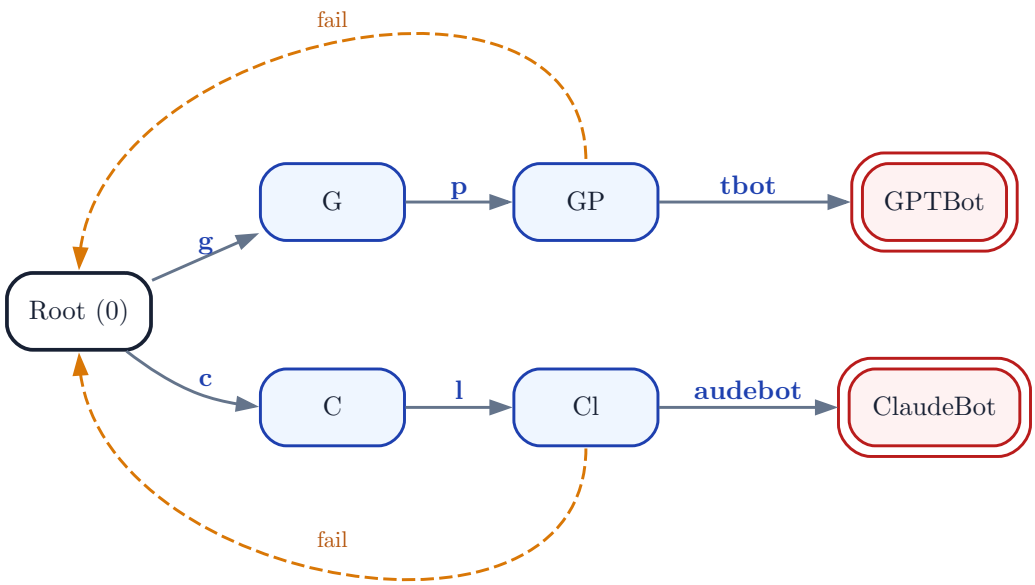
सघन DFA के रूप में Aho-Corasick

Sibuna के पास लिटरल पैटर्न के दो समुच्चय हैं: लगभग चालीस बॉट सिग्नेचर और लगभग सौ आक्रमण सिग्नेचर। किसी फ़ील्ड को N पैटर्न से एक-एक करके जाँचना $O(NM)$ है; Aho और Corasick (1975) का ऑटोमेटन failure link वाला trie बनाने के बाद उसे एक बार में $O(M)$ में स्कैन करता है। Sibuna बिल्ड के समय failure link को ट्रांज़िशन टेबल में समाहित कर देता है, इसलिए स्कैन करना प्रति इनपुट बाइट एक टेबल लोड है, और पैटर्न समुच्चय पर कोई ब्रांच नहीं होती। ऑटोमेटन `sibuna-patterns` क्रेट में है:

```
/// `state` (एक पंक्ति ऑफ़सेट) से `byte` पढ़कर आगे बढ़ता है।
fn step(&self, state: usize, byte: u8) -> usize {
    self.table[state + usize::from(self.classes[usize::from(byte)])] as usize
}

pub fn find_first(&self, haystack: &[u8]) -> Option<Match> {
    let mut state = 0;
    for (index, &byte) in haystack.iter().enumerate() {
        state = self.step(state, byte);
        if state >= self.first_match {
            let id = self.first_output_in(self.match_state(state),
self.all_tags)?;
            return Some(self.report(id, index + 1));
        }
    }
    None
}
```

एक अवस्था को एक सपाट टेबल में उसकी पंक्ति के ऑफ़सेट के रूप में रखा जाता है, इसलिए एक कदम एक जोड़ और एक लोड है। अवस्थाओं को इस तरह क्रमांकित किया जाता है कि हर मैच अवस्था बाकी सभी अवस्थाओं के बाद आए; तब मैच का पता लगाना `first_match` से एक तुलना भर है, और भीतरी लूप में कोई दूसरी टेबल लोड नहीं करनी पड़ती।



चित्र 9 दो सिग्नेचर के लिए trie अवस्थाएँ और failure edge

ऑटोमेटन का अवस्था बजट एक रन-टाइम विकल्प है। नीति इंजन अधिकतम एक रूट और सभी सिग्नेचर की लंबाइयों का योग आरक्षित करता है, जो trie अवस्थाओं की एक सुरक्षित ऊपरी सीमा है। इससे ऐतिहासिक रूप से बहुत बड़ी निश्चित टेबल से बचा जाता है, और हर byte class के लिए सघन ट्रांज़िशन बना रहता है। हर पैटर्न एक छोटा श्रेणी टैग रखता है (0 से 63, ताकि टैग का समुच्चय एक 64-बिट शब्द हो)। case folding ऑटोमेटन के साथ बनी 256 प्रविष्टियों वाली byte-class टेबल का उपयोग करता है: किसी अक्षर के दोनों case एक class साझा करते हैं, और जो बाइट किसी पैटर्न में नहीं आते वे एक class साझा करते हैं, इसलिए टेबल की एक पंक्ति 256 के बजाय अलग-अलग पैटर्न बाइट की संख्या से एक अधिक चौड़ी होती है। इंजन के सटीक आकार और अनुक्रमिक सबस्ट्रिंग खोज से तुलना भाग VIII में दर्ज है। सबस्ट्रिंग खोज वही पैटर्न और any-match अर्थ (semantics) उपयोग करती है।

रेगुलर एक्सप्रेशन, SIMD इंजन और चुनाव

सिग्नेचर चरण के लिए तीन परिवारों को तौला गया:

- बैकट्रैकिंग रेगुलर एक्सप्रेशन (बैकट्रैकिंग नियम-समुच्चय की शैली) प्रति अनुरोध सैकड़ों पैटर्न चलाते हैं और प्रति मैच आवंटन करते हैं; ये सबसे धीमा विकल्प हैं और false positive की सबसे अधिक संभावना इन्हीं में है।
- *SIMD* लिटरल इंजन (Hyperscan, Vectorscan) shuffle पर आधारित nibble मास्क ("Teddy" एल्गोरिदम) से 16 से 32 बाइट के ब्लॉक को पहले छानते हैं और उम्मीदवारों का सत्यापन ऑटोमेटा से करते हैं। बड़े लिटरल समुच्चय वाले लंबे इनपुट पर ये बेहतरीन हैं। इसकी कीमत है कई मेगाबाइट की C++ निर्भरता, कंपाइल-टाइम पैटर्न डेटाबेस, कोई WebAssembly टारगेट नहीं, और सौ बाइट के इनपुट के लिए कोई लाभ नहीं: पूरा फ़ील्ड एक cache miss से कम समय में स्कैन हो जाता है।
- सघन *Aho-Corasick* और छोटे संरचनात्मक टोकनाइज़र, libinjection की परंपरा, एक निश्चित टेबल में सटीक रैखिक-समय व्यवहार देते हैं और बाइनरी के बाहर किसी चीज़ की ज़रूरत नहीं रखते।

Sibuna के फ़ील्ड छोटे हैं, इसलिए तीसरा परिवार मेमोरी और गति दोनों में जीतता है; भाग VIII 8 KiB बॉडी पर लगभग 1.56 ns प्रति बाइट दर्ज करता है (सिमैंटिक स्कैन में 12,784.3 ns लगते हैं, `benchmarks/results/primitives-latest.md` में `8kb_body_semantic_scan`)। यही एक वर्कलोड है जहाँ SIMD प्रीफ़िल्टर लाभ देगा, और SID 0006 इसे एक खुले बिंदु के रूप में रखता है।

IPv4 और IPv6 के लिए radix trie

उद्देश्य 128-बिट trie में longest-prefix लुकअप को ट्रेस करें, IPv4 रूट का शॉर्टकट समझाएँ, और देखें कि trie क्लस्टर से भरे जाने वाले प्रतिष्ठा स्टोर का काम भी कैसे करता है।

128-बिट कुंजियों पर एक बाइनरी trie दोनों परिवारों की सेवा करता है: IPv6 प्रीफ़िक्स जैसे के तैसे डाले जाते हैं, IPv4 प्रीफ़िक्स `::ffff:0:0/96` में मैप किए जाते हैं, इसलिए एक `/24` गहराई 120 का पाथ बन जाता है। नोड `u32` से इंडेक्स किए गए एक सपाट `Vec` में रहते हैं, जिसमें इंडेक्स शून्य रूट भी है और "कोई चाइल्ड नहीं" भी, इसलिए एक लुकअप अधिकतम 128 आश्रित लोड है, बिना pointer chasing के। यह trie `sibuna-cidr` क्रेट का `PrefixTrie` है।
हर IPv4 लुकअप के लिए मैप किए गए प्रीफ़िक्स के 96 स्तरों पर चलना, IPv4 पते के पहले बिट तक पहुँचने से पहले ही 96 आश्रित लोड खर्च करेगा। इसलिए trie प्रारंभ होते समय गहराई 96 वाले नोड को दर्ज कर लेता है और IPv4 लुकअप वहीं से शुरू करता है:

```
let (mut node, mut depth, mut best) = if is_ipv4_mapped(key) {
    let start = self.ipv4_root;
    let value = self.nodes[start as usize].value.as_ref();
    (start, IPV4_DEPTH, value.or(self.nodes[0].value.as_ref()))
} else {
    (0, 0, self.nodes[0].value.as_ref())
};
```

IPv4 पते के वर्गीकरण में 32.0 ns और IPv6 पते में 79.3 ns लगते हैं (`benchmarks/results/primitives-latest.md` में `ipv4_cidr_classification` और `ipv6_cidr_classification`)। इंजन trie से दो बार परामर्श करता है: नियम टेबल से पहले अंतिम `allow`/`deny` निर्णयों के लिए, ताकि क्लस्टर-व्यापी प्रतिबंध हर नियम पर भारी पड़े, और उसके बाद `challenge` निर्णयों के लिए।

सिमैंटिक फ़ायरवॉल

उद्देश्य byte-class प्री-स्कैन, टैग किया गया सिग्नेचर ऑटोमेटन, tautology पहचानने वाला SQL टोकनाइज़र, HTML और shell संरचना की जाँचें, canonicalisation, और false positive का वह अनुशासन समझें जिसके तहत किसी वास्तविक ब्राउज़र अनुरोध को बिना छेड़े पास होना चाहिए।

False positive से एक सबक

WAF का पहला संस्करण 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 को SQL injection के रूप में वर्गीकृत किया गया और उसे `403` मिला। इस सबक ने डिज़ाइन को आकार दिया: कोई डिटेक्टर केवल संरचना पर सक्रिय हो सकता है, और परीक्षण सूट अब एक पूरा Chrome अनुरोध, जिसमें `select`, `--` और एक एपॉस्ट्रॉफ़ी वाली कमेंट बॉडी शामिल है, इंजन से गुज़ारता है और पुष्टि करता है कि वह साफ़ निकलता है।

एक पास में byte class

हर निरीक्षित फ़ील्ड को एक बार स्कैन करके यह इकट्ठा किया जाता है कि कौन-सी byte class आती हैं: quote, `=`, `<`, shell separator, `%`, NUL, canonicalisation के ट्रिगर, और दोहरे स्पेस। हर डिटेक्टर दोबारा स्कैन करने के बजाय इन class से परामर्श करता है, और canonicalised दूसरा पास तभी चलता है जब कोई ट्रिगर दिखा हो। स्कैन प्रति

बाइट 256 प्रविष्टियों वाली टेबल का एक लुकअप है, sibuna-policy के WAF मॉड्यूल से। चार accumulator OR शृंखलाओं को छोटा रखते हैं, और दो बाइट-जोड़ों (/* और दो सटे हुए स्पेस) को तभी खोजा जाता है जब किसी अकेले बाइट ने पहले ही canonicalisation की माँग न की हो:

```
pub(crate) fn scan(text: &[u8]) -> Classes {
    let (chunks, remainder) = text.as_chunks::<8>();
    // चार accumulator OR की शृंखलाएँ छोटी रखते हैं, ताकि लोड एक के बाद एक जारी हो सकें।
    let mut lanes = [0u16; 4];
    for chunk in chunks {
        lanes[0] |= TABLE[usize::from(chunk[0])] | TABLE[usize::from(chunk[4])];
        lanes[1] |= TABLE[usize::from(chunk[1])] | TABLE[usize::from(chunk[5])];
        lanes[2] |= TABLE[usize::from(chunk[2])] | TABLE[usize::from(chunk[6])];
        lanes[3] |= TABLE[usize::from(chunk[3])] | TABLE[usize::from(chunk[7])];
    }
    let mut bits = lanes[0] | lanes[1] | lanes[2] | lanes[3];
    for &byte in remainder {
        bits |= TABLE[usize::from(byte)];
    }
    // बाइट-जोड़े तभी तक मायने रखते हैं जब तक किसी अकेले बाइट ने canonicalisation की माँग न की हो।
    if bits & CANONICAL == 0
        && ((bits & SPACE != 0 && has_pair(text, b' ', b' '))
            || (bits & (SLASH | STAR) == SLASH | STAR && has_pair(text, b'/',
b'*')))
    {
        bits |= CANONICAL;
    }
    Classes { bits }
}
```

डिटेक्टर

- सिग्नेचर। हर श्रेणी के सभी मज़बूत सिग्नेचर एक टैग किए गए ऑटोमेटन में रहते हैं; हिट एक अंतिम deny है, जिसका नाम waf:sqli, waf:xss, waf:path-traversal या waf:rce होता है।
- *SQL injection*। एक-पास वाला शब्द स्कैनर हर अल्फ़ान्यूमेरिक शब्द (अधिकतम सात बाइट) को कीवर्ड टेबल में खोजता है और or / and पर literal = literal tautology की जाँच करता है; id=1 or 1=1 अंतिम है। अन्यथा quote बाइट अनिवार्य है और साक्ष्य का स्कोर इनमें से कम से कम 4 होना चाहिए: quote (1), अधिकतम दो कीवर्ड (2), कमेंट मार्कर (1), एक = (1)। admin'-- अपने आप में अंतिम है। "it's a group order from the shop" का स्कोर 3 है और यह पास हो जाता है।
- *Cross-site scripting*। < आवश्यक है; यह तब सक्रिय होता है जब स्क्रिप्ट रख सकने वाला कोई टैग एट्रिब्यूट की स्थिति में on<event>= एट्रिब्यूट के साथ आता है। onboarding=1 नहीं गिना जाता।
- *Command injection*। एक separator आवश्यक है; यह तब सक्रिय होता है जब ;, |, &&, $(, backtick या newline के बाद कोई ज्ञात कमांड नाम आता है। status ok; done | next पास हो जाता है।
- *Path traversal*। सिग्नेचर (../, ..%2f, /etc/passwd, id_rsa, ...) और साथ में NUL तथा %00।

जिन संरचनात्मक हेडर के व्याकरण में वैध रूप से quote और तारांकन होते हैं (Accept*, Content-Type, Sec-*, If-*, ...), उन्हें छोड़ दिया जाता है; पाथ, क्वेरी, User-Agent, अन्य हेडर और बॉडी के पहले 8 KB का निरीक्षण होता है, और जिन फ़ील्ड में percent escape, plus चिह्न, कमेंट ओपनर या समेटे जा सकने वाले whitespace हों, उन्हें 8 KB के स्टैक बफ़र में canonicalise करके दोबारा निरीक्षित किया जाता है, ताकि 1%27/**/UnIoN/**/SeLeCt और %252e%252e/ कच्चे सिग्नेचर पर सिमट जाएँ।

लागत

निरीक्षण की लागत इंजन को दिए गए फ़ील्ड और बाइट पर निर्भर करती है, केवल अनुरोधों की संख्या पर नहीं। बेंचमार्क अध्याय Gate, Shield और 8 KB बॉडी की स्थितियाँ अलग-अलग रिपोर्ट करता है। 8 KB की सीमा निरीक्षण की सीमा है, यह दावा नहीं कि अपलोड का शेष भाग सुरक्षित है। forward-auth का उपयोग करने वाले इनग्रेस को अनुरोध की प्रासंगिक जानकारी आगे भेजनी चाहिए; छोड़े गए बाइट का निरीक्षण यह प्रोसेस नहीं कर सकता।

घोषणात्मक नीति इंजन

> **उद्देश्य** नियम मॉडल, मूल्यांकन का क्रम, थ्रेशोल्ड के साथ WEIGH स्कोरिंग, और JSON नीति का व्याकरण पढ़ें।

नियम वैकल्पिक मानदंडों (पाथ पैटर्न, User-Agent पैटर्न, अधिकतम चार हेडर, अधिकतम आठ IPv4/IPv6 CIDR ब्लॉक, एक JA4 फ़िंगरप्रिंट या प्रीफ़िक्स) का संयोजन (AND) है, जिसमें एक एक्शन `ALLOW`, `DENY`, `CHALLENGE` या `WEIGH`, वर्क बिट्स में वैकल्पिक प्रति-नियम कठिनाई और एल्गोरिदम, और एक चिह्नित `weight` होता है। मूल्यांकन का क्रम:

1. सिमैंटिक WAF (केवल Shield): उल्लंघन एक अंतिम `deny` है।
2. प्रतिष्ठा trie: `deny` या `allow` प्रीफ़िक्स अंतिम हैं।
3. क्रम से नियम: पहला अंतिम मैच लौटता है; `WEIGH` मैच जुड़ते जाते हैं।
4. स्कोर: ऋणात्मक योग `allow`; ≥ 40 योग `deny`; ≥ 10 योग `challenge`, थ्रेशोल्ड से ऊपर हर 5 अंक पर एक अतिरिक्त वर्क बिट के साथ, अधिकतम छह।
5. स्टैटिक बायपास पाथ, फिर trie के `challenge` निर्णय, फिर बॉट ऑटोमेटन।
6. डिफ़ॉल्ट एक्शन, `challenge`, जब तक नीति फ़ाइल कुछ और न कहे।

बिल्ट-इन नियम `/.well-known/*`, `/favicon.ico`, `/robots.txt` और `/__sibuna/*` की अनुमति देते हैं, `CF-Worker` क्लाइंट और `Amazonbot` को अस्वीकार करते हैं, और हर `Mozilla` User-Agent को चैलेंज करते हैं: ब्राउज़र स्ट्रिंग को जाली बनाना मुफ़्त है, इसलिए उसे प्रवेश देने से गेट केवल सजावटी रह जाएगा।

```
{
  "default_action": "CHALLENGE",
  "waf": true,
  "thresholds": { "challenge_at": 10, "deny_at": 40, "bits_step": 5 },
  "ip_rules": { "10.0.0.0/8": "ALLOW", "2001:db8::/32": "DENY" },
  "rules": [
    { "name": "internal-api", "path": "/api/*", "headers": { "X-Api-Key": ".*" },
    "action": "ALLOW" },
    { "name": "checkout", "path": "/checkout/*", "action": "CHALLENGE",
      "challenge": { "difficulty": 20, "algorithm": "posw" } },
    { "name": "headless", "user_agent": "Headless", "action": "WEIGH", "weight":
30 },
    { "name": "no-language", "headers": { "Accept-Language": "" },
    "action": "WEIGH", "weight": -5 }
  ]
}
```

हल किया गया ट्रेस: वैध सत्र के साथ एन्कोड किया गया आक्रमण

वैध सत्र रखने वाले क्लाइंट से `GET /search?q=%2527%25200R%25201%253D1` पर विचार करें। पहले HTTP पार्सर पाथ और क्वेरी को उनके बाइट कॉपी किए बिना अलग करता है। स्थानीय सीमाएँ चलती हैं, फिर Shield अनुरोध का

निरीक्षण करता है। percent चिह्न फ़ील्ड को canonicalization के लिए चिह्नित करता है। डिकोडिंग का एक पास `%27%20OR%201%3D1` प्रकट करता है; दूसरा `' OR 1=1` प्रकट करता है।

अब टोकनाइज़र एक quote, कीवर्ड `OR` और बराबर लिटरल ऑपरेंड देखता है। अस्वीकृति एक अंतिम नीति परिणाम है। वैध सत्र इसे अनुमति में नहीं बदलता: सत्र जाँच केवल चैलेंज निर्णय का निपटारा करती है। स्टोरेज हुक एक सीमित घटना रिकॉर्ड कॉपी करता है; SQL बाद में चलता है। जब ऑरिजिन कनेक्शन निष्क्रिय हो, तब रिस्पॉन्स पाथ को इंजन स्नैपशॉट पकड़े रखने की ज़रूरत नहीं होती।

अभ्यास 6.1. पेलोड को `O'Reilly` से और फिर `order=1` से बदलें। इनमें से कोई भी अकेले SQL injection क्यों स्थापित न करे? एक ऐसे false positive का उदाहरण दें जो केवल सबस्ट्रिंग पर आधारित डिटेक्टर उत्पन्न कर सकता है।

अभ्यास 6.2. LDAP injection (जैसे `)(|(` पैटर्न) के लिए डिटेक्टर जोड़ने हेतु आवश्यक byte-class टेबल प्रविष्टियाँ लिखें, और समझाएँ कि वह कौन-से मौजूदा class बिट दोबारा उपयोग कर सकता है।

अपरिवर्तनीय नियम समझाएँ. किसी समीक्षक को समझाएँ कि `Accept: */*` false positive क्यों था, नया SQL डिटेक्टर किस संरचनात्मक गुण की माँग करता है, और ऑटोमेटन तथा टोकनाइज़र कार्य को कैसे बाँटते हैं।

भाग VII

ब्राउज़र का proof-of-work इंजन

हम उस *WebAssembly* मॉड्यूल को देखते हैं जो सर्वर के अपने सत्यापनकर्ता क्रेट को कंपाइल करता है, *Web Worker* प्रोटोकॉल को, *JavaScript* फ़ॉलबैक प्रूवर को, और इंटरस्टिशियल को।

एक स्रोत, दो टारगेट

उद्देश्य इस अध्याय के अंत तक हमें यह कर पाना चाहिए: समझाना कि ब्राउज़र मॉड्यूल सर्वर के अपने `sibuna-pow` क्रेट से कैसे बनता है, एक्सपोर्ट किए गए फ़ंक्शन के नाम बताना, और यह वर्णन करना कि अनुक्रमिक कार्य के प्रूवर को किसी टैब में कितनी मेमोरी चाहिए।

बिल्ड की व्यवस्था

ब्राउज़र सॉल्वर ठीक उन्हीं स्रोतों को कंपाइल करता है जिनसे सर्वर सत्यापन करता है। `cmd/sibuna-pow-wasm` `cdylib` प्रकार का एक `no_std` लाइब्रेरी क्रेट है, जो `wasm32-unknown-unknown` के लिए बनता है, और जिसकी निर्भरताएँ केवल वर्कस्पेस के क्रेट `sibuna-pow` और `sibuna-crypto` हैं:

```
# cmd/sibuna-pow-wasm/Cargo.toml
[lib]
crate-type = ["cdylib", "rlib"]

[dependencies]
sibuna-crypto = { workspace = true }
sibuna-pow = { workspace = true }

# Cargo.toml (वर्कस्पेस रूट): मॉड्यूल छोटे आकार के लिए बिल्ड होता है
[profile.release.package.sibuna-pow-wasm]
opt-level = "z"
debug = false
strip = true
```

डेमन इस मॉड्यूल को एम्बेड करता है। उसकी बिल्ड स्क्रिप्ट (`cmd/sibuna/build.rs`) वर्कस्पेस की टारगेट डायरेक्टरी (`target/wasm32-unknown-unknown/wasm/`) में एक नेस्टेड `cargo rustc -p sibuna-pow-wasm --target wasm32-unknown-unknown --profile wasm` चलाती है। नेस्टेड बिल्ड केवल `wasm` प्रोफ़ाइल की डायरेक्टरी लॉक करता है, इसलिए वह कभी बाहरी बिल्ड की प्रतीक्षा नहीं करता। अंतिम लिंक को `--compress-relocations` मिलता है। स्क्रिप्ट जाँचती है कि परिणाम WebAssembly के magic बाइट से शुरू होता है, और उसे बिल्ड आउटपुट डायरेक्टरी में कॉपी करती है। WebAssembly टारगेट के बिना बिल्ड के लिए `SIBUNA_POW_WASM=<path>` इसके बजाय पहले से बना मॉड्यूल एम्बेड करता है। फिर डेमन इन बाइट को अपनी बाइनरी में शामिल करता है:

```
/// ब्राउज़र सॉल्वर, जिसे `build.rs` `cmd/sibuna-pow-wasm` से बिल्ड करता है।
pub const WASM_SOLVER: &[u8] =
    include_bytes!(concat!(env!("OUT_DIR"), "/sibuna-pow.wasm"));
```

```
/// सॉल्वर का Web Worker।
pub const WORKER_JS: &str = include_str!("../../../web/worker.js");
```

कोई दूसरा इम्प्लीमेंटेशन है ही नहीं जो धीरे-धीरे अलग हो जाए। मॉड्यूल डेमन से `/__sibuna/wasm/sibuna-pow.wasm` पर एक घंटे के cache हेडर (`Cache-Control: public, max-age=3600`) के साथ दिया जाता है। ऊपर की प्रोफ़ाइल से बनने पर दोनों टियर के साथ मॉड्यूल का आकार 14,744 बाइट है (`benchmarks/results/primitives-latest.json` में `wasm_bytes`)।

एक्सपोर्ट

मॉड्यूल छह फ़ंक्शन एक्सपोर्ट करता है, और वर्कर स्क्रिप्ट उन्हें नाम से कॉल करती है:

एक्सपोर्ट	काम
`sibuna_get_buffer_ptr()` / `sibuna_get_buffer_len()`	256-बाइट इनपुट बफ़र, जिसमें वर्कर चैलेंज स्ट्रिंग लिखता है
`sibuna_solve_step(ptr, len, bits, start, max_steps)`	Hashcash: `start` से `max_steps` नॉनस खोजता है; नॉनस या all-ones सेंटिनल लौटाता है
`sibuna_solve_sha256(ptr, len, bits)`	नॉनस 0 से एक बैच में Hashcash (`u32::MAX` कदमों के साथ `sibuna_solve_step`)
`sibuna_posw_solve(len, depth, challenges)`	अनुक्रमिक कार्य: बफ़र पर प्रूवर चलाता है और प्रमाण की लंबाई लौटाता है, या अमान्य आर्ग्युमेंट के लिए 0
`sibuna_posw_proof_ptr()`	प्रूवर वर्कस्पेस में प्रमाण के बाइट की शुरुआत

प्रूवर का वर्कस्पेस, यानी रखे गए ऊपरी स्तर, सहोदर स्टैक और प्रमाण, लगभग 92 KiB का एक स्टैटिक क्षेत्र है; कोई एलोकेटर लिंक नहीं होता, और मॉड्यूल में कोई import नहीं है। `Workspace::new` एक `const fn` है, इसलिए स्टैटिक की लागत केवल शून्य की गई मेमोरी है, मॉड्यूल में प्रारंभ किया गया डेटा नहीं।

> **इम्प्लीमेंटेशन: `sibuna_solve_step`** Hashcash एक्सपोर्ट: linear memory से चैलेंज उधार लेता है और साझा नॉनस खोज का एक बैच चलाता है। `cmd/sibuna-pow-wasm/src/lib.rs` में (क्रेट `sibuna-pow-wasm`)।

```
#[unsafe(no_mangle)]
pub unsafe extern "C" fn sibuna_solve_step(
    challenge_ptr: *const u8,
    challenge_len: usize,
    difficulty_bits: u32,
    start_nonce: u64,
    max_steps: u32,
) -> u64 {
    let challenge: &[u8] = if challenge_len == 0 {
        &[]
    } else {
        // SAFETY: कॉल करने वाला गारंटी देता है कि `challenge_ptr` पर `challenge_len`
        // पढ़ने योग्य बाइट हैं, जो कॉल के पूरे समय नहीं बदलते।
        unsafe { core::slice::from_raw_parts(challenge_ptr, challenge_len) }
    };
```

```
    Search::new(challenge)
        .find(difficulty_bits, start_nonce, u64::from(max_steps))
        .unwrap_or(EXHAUSTED)
}
```

एक्सपोर्ट क्रेट की foreign function सीमा हैं, और वही एकमात्र स्थान है जहाँ उसका `unsafe` कोड रहता है। इनपुट बफ़र और प्रूवर वर्कस्पेस एक `ExportCell` में स्टैटिक हैं, जो `UnsafeCell` के चारों ओर एक रैपर है और एक बताए गए कारण के साथ `Sync` घोषित है: shared memory के बिना WebAssembly इंस्टेंस एक थ्रेड चलाता है, और हर एक्सपोर्ट लौटने से पहले अपने संदर्भ बनाता है, उनका उपयोग करता है और उन्हें छोड़ देता है, इसलिए किसी सेल में एक समय पर अधिकतम एक ही संदर्भ जीवित होता है। होस्ट पर, जहाँ परीक्षण एक्सपोर्ट को सीधे कॉल करते हैं, केवल परीक्षण के लिए बना एक mutex उन्हें क्रम में लगाता है। panic संदेश फ़ॉर्मेट करने के बजाय trap करता है (`unreachable` निर्देश), जिससे मॉड्यूल फ़ॉर्मेटिंग कोड और import से मुक्त रहता है; panic तक पहुँचना बनावट से ही असंभव है, क्योंकि `sibuna-pow` के फ़ंक्शन total हैं और कभी panic नहीं करते।

प्रीफ़िक्स की पूर्व-हैशिंग

Hashcash का भीतरी लूप चैलेंज और कोलन को एक बार सोख लेता है और प्रति नॉनस SHA-256 अवस्था की प्रति (clone) बनाता है, इसलिए हर उम्मीदवार की लागत दो के बजाय एक compression है। खोज `sibuna_pow::hashcash::Search` है, जिसे ब्राउज़र मॉड्यूल और नेटिव संदर्भ सॉल्वर साझा करते हैं:

```
pub fn new(challenge: &[u8]) -> Self {
    let mut prefix = Sha256::new();
    prefix.update(challenge);
    prefix.update(b":");
    Self { prefix }
}

pub fn digest(&self, nonce: u64) -> Digest {
    let mut buf = [0u8; NONCE_DIGITS_MAX];
    let mut hasher = self.prefix.clone(); // सोखे गए प्रीफ़िक्स की value copy
    hasher.update(format_decimal(nonce, &mut buf));
    hasher.finalize()
}

pub fn find(&self, bits: u32, start: u64, steps: u64) -> Option<u64> {
    let mut nonce = start;
    for _ in 0..steps {
        if meets_bits(&self.digest(nonce), bits) {
            return Some(nonce);
        }
        nonce = nonce.wrapping_add(1);
    }
    None
}
```

hasher की अवस्था लगभग 110 बाइट का एक साधारण मान है, इसलिए clone सस्ता है। अनुक्रमिक कार्य का प्रूवर भी यही तरकीब उपयोग करता है, जिसमें स्टेटमेंट χ को `base_hasher` में एक बार सोखा जाता है और हर नोड के लिए clone किया जाता है, और अब सत्यापनकर्ता भी ऐसा करता है: वह हर चैलेंज की गई लीफ़ को उस अवस्था से निकालता है जिसने स्टेटमेंट को एक बार सोख लिया है, जिससे समान डाइजेस्ट के साथ सत्यापन t compression सस्ता हो जाता है।

सर्वर पर compression फ़ंक्शन x86-64 SHA extension या ARMv8 SHA-2 निर्देशों का उपयोग करता है, जिनका पता रन टाइम पर लगाया जाता है। ब्राउज़र में ऐसे कोई निर्देश नहीं होते, और सॉल्वर लगभग सारा समय पोर्टेबल

compression फ़ंक्शन (`crates/crypto/src/sha256/portable.rs`) में बिताता है। वह फ़ंक्शन WebAssembly के लिए लिखा गया है: 64 राउंड unrolled हैं, message schedule 16 शब्दों की एक विंडो में रहता है जिसमें 16 से आगे का हर राउंड उस शब्द को अधिलेखित करता है जिसकी उसे अब ज़रूरत नहीं, और आठ कार्यकारी चरों को खिसकाने के बजाय राउंड दर राउंड नया नाम दिया जाता है:

```
fn compress_block(state: &mut State, block: &Block) {
    let mut w = [0u32; 16];
    for (word, bytes) in w.iter_mut().zip(block.as_chunks::<4>().0) {
        *word = u32::from_be_bytes(*bytes);
    }
    let [mut a, mut b, mut c, mut d, mut e, mut f, mut g, mut h] = *state;
    eight_rounds!(w, 0, a, b, c, d, e, f, g, h);
    eight_rounds!(w, 8, a, b, c, d, e, f, g, h);
    // ... राउंड 16 से 55
    eight_rounds!(w, 56, a, b, c, d, e, f, g, h);
    for (word, value) in state.iter_mut().zip([a, b, c, d, e, f, g, h]) {
        *word = word.wrapping_add(value);
    }
}

/// schedule शब्द W(i) लौटाता है; राउंड 16 और आगे के लिए उसकी गणना उसी जगह करता है।
#[inline(always)]
fn schedule(w: &mut [u32; 16], i: usize) -> u32 {
    if i >= 16 {
        let w15 = w[(i + 1) % 16];
        let w2 = w[(i + 14) % 16];
        let s0 = w15.rotate_right(7) ^ w15.rotate_right(18) ^ (w15 >> 3);
        let s1 = w2.rotate_right(17) ^ w2.rotate_right(19) ^ (w2 >> 10);
        w[i % 16] = w[i % 16]
            .wrapping_add(s0)
            .wrapping_add(w[(i + 9) % 16])
            .wrapping_add(s1);
    }
    w[i % 16]
}
```

हर `eight_rounds!` चरों के नामों का एक पूरा चक्कर है: मैक्रो उन्हें प्रति राउंड एक स्थान घुमाकर देता है, इसलिए एक राउंड उनमें से केवल दो लिखता है और बाकी छह को कभी नहीं हिलाता। लिटरल राउंड इंडेक्स के साथ हर schedule स्थान एक स्थिरांक है, और पूरा ब्लॉक सीधी रेखा वाला कोड बन जाता है जिसके मान WebAssembly locals में रहते हैं। यह लेआउट V8 में rolled लूप वाले 64-शब्द schedule से लगभग 20% तेज़ चलता है, और नेटिव रूप से लगभग 14% तेज़ (Rust इम्प्लीमेंटेशन पर मापा गया, अक्टूबर 2026)।

वर्कर प्रोटोकॉल

> **उद्देश्य** संदेश प्रोटोकॉल पढ़ें, फ़ॉलबैक का क्रम समझें, और वह सेंटिनल बग देखें जो WebAssembly और JavaScript के बीच signed-integer सीमा के कारण हुआ।

पेज `web/worker.js` को एक संदेश पोस्ट करता है और प्रगति, फ़ॉलबैक और परिणाम के संदेश प्राप्त करता है:

```
// पेज -> वर्कर
{ challenge: spec.id, algorithm: spec.algorithm, difficulty: spec.difficulty,
    challenges: spec.challenges }

// वर्कर -> पेज
{ type: 'progress', iterations: 40000 }
{ type: 'fallback', message: 'Wasm unavailable; running JS prover' }
{ type: 'solved', challenge: spec.id, solution: { nonce: '90766' } }  // hashcash
{ type: 'solved', challenge: spec.id, solution: { proof: '<base64url>' } } // posw
{ type: 'error', message: '...' }
```

वर्कर पहले WebAssembly आज़माता है और जब `WebAssembly` अनुपस्थित हो, नीति द्वारा ब्लॉक हो, या इंस्टेंशिएट न हो पाए, तब शुद्ध JavaScript पर लौट आता है। JavaScript प्रूवर `sibuna-pow` के Rust प्रूवर का लेबल दर लेबल अनुकरण करते हैं और *byte-identical* आउटपुट देते हैं। परीक्षण दोनों टियर के लिए निश्चित नॉनस और प्रमाण तय करते हैं, और SID 0010 की माँग है कि सॉल्वर headless ब्राउज़र में WebAssembly के साथ और JavaScript फ़ॉलबैक के साथ, दोनों तरह पास हो। JavaScript प्रूवर बहुत धीमे हैं, लेकिन वे गारंटी देते हैं कि हर ब्राउज़र पास हो सकता है।

signed सेंटिनल WebAssembly 64-बिट पूर्णांक को JavaScript को एक *signed* `BigInt` के रूप में लौटाता है। Hashcash एक्सपोर्ट "बैच समाप्त" का संकेत सभी बिट एक (`EXHAUSTED`, `u64::MAX`) से देता है, जिसे JavaScript `-1n` के रूप में देखता है, `18446744073709551615n` के रूप में नहीं। मूल वर्कर unsigned मान से तुलना करता था, इसलिए लूप पहले बैच के बाद एक नकली नॉनस के साथ बाहर निकल जाता था। सुधार एक कॉल है, `BigInt.asUintN(64, result)`, और वर्कर इसे बनाए रखता है:

```
// WebAssembly के i64 परिणाम signed BigInt के रूप में आते हैं, इसलिए u64 का
// "exhausted" सेंटिनल (सभी बिट एक) -1n के रूप में पढ़ा जाता है।
const found = BigInt.asUintN(64,
    ex.sibuna_solve_step(ptr, encoded.length, bits, nonce, maxSteps));
if (found !== 18446744073709551615n) return found.toString();
```

यह सबक व्यापक है: सीमा पार करने वाले हर u64 को सामान्यीकृत करना होगा। Rust इसे JavaScript की ओर लागू नहीं कर सकता; एक्सपोर्ट का प्रकार `u64` कहता है, और रूपांतरण इंजन में होता है।

कैलिब्रेशन

उद्देश्य कठिनाई की सेटिंग को ब्राउज़र इंजन में और नेटिव सिलिकॉन पर वास्तविक समय (wall-clock time) से जोड़ें।

किसी एक लैपटॉप के औसत हल से कठिनाई न चुनें। Hashcash की ज्यामितीय टेल होती है; ब्राउज़र के समय में शेड्यूलिंग, तापीय अवस्था, स्टार्टअप और कंपाइलेशन शामिल होते हैं। डिप्लॉयमेंट के कैलिब्रेशन में ब्राउज़र संस्करण, डिवाइस, एल्गोरिदम, पैरामीटर और बार-बार किए गए हलों का वितरण दर्ज होना चाहिए। मॉड्यूल लाने और कंपाइल करने को स्थिर अवस्था की हैशिंग से अलग रखें।

हल किया गया उदाहरण: कैलिब्रेशन योजना b को स्थिर रखें, मॉड्यूल को warm करें, और लक्षित हर डिवाइस वर्ग पर स्वतंत्र रूप से जारी कम से कम 100 पहेलियाँ हल करें। माध्यिका और ऊँचे पर्सेंटाइल वाली अवधियाँ, विफलताएँ

और प्रमाण के आकार दर्ज करें। $b+1$ पर दोहराएँ। अपेक्षित प्रयास संख्या दोगुनी होती है; किसी एक नमूने का दोगुना होना ज़रूरी नहीं। धीमे डिवाइस के बजट के आधार पर सेटिंग चुनें, फिर लाने, हल करने और सत्यापन के पूरे प्रवाह का परीक्षण करें।

रिपॉज़िटरी के नेटिव प्रिमिटिव मापन किसी फ़ोन या ब्राउज़र पर वादा किए गए हल-समय का साक्ष्य नहीं हैं। एक WASM मॉड्यूल वही अंकगणित चला सकता है जबकि उसकी स्टार्टअप, निष्पादन और मेमोरी लागतें अलग हों: नेटिव रूप से सर्वर SHA-NI से हैश करता है, जबकि ब्राउज़र ऊपर वाला पोर्टेबल compression फ़ंक्शन चलाता है। वर्कर के संदेश वह सीमा बनाते हैं जहाँ इन लागतों को सर्वर के अनुरोध हैंडलर में टाइमर जोड़े बिना मापा जा सकता है।

इंटरस्टिशियल

पेज (`web/challenge.html`, डेमन में एम्बेडेड) एक अकेली HTML फ़ाइल है जिसमें एक कार्ड, एक अनिश्चित प्रगति पट्टी और एक स्टेटस पंक्ति है। यह अपने ही स्थान (`?path=`) के लिए चैलेंज माँगता है, वर्कर शुरू करता है, `fetch` से समाधान पोस्ट करता है, और सफलता पर दोबारा लोड होता है; `fetch` कॉल jitter वाले घातीय backoff के साथ रीट्राई करते हैं, और `4xx` सत्यापन उत्तर उसके निदान शीर्षक के साथ दिखाया जाता है। पेज की `Content-Security-Policy` ठीक एक स्क्रिप्ट, सॉल्वर ब्लॉक, को उसके SHA-256 हैश से अनुमति देती है, जिसकी गणना डेमन एम्बेडेड पेज से एक बार करता है। स्क्रीन से बाहर रखा एक हनीपॉट लिंक (`/__sibuna/honeypot`) लोगों और स्क्रिप्ट चलाने वाले ब्राउज़र के लिए अदृश्य है; हर `href` का अनुसरण करने वाले अंधे क्रॉलर स्वयं पर प्रतिबंध लगवा लेते हैं।

अभ्यास 7.1. मान लें कि WebAssembly प्रूवर किसी लैपटॉप पर डिफ़ॉल्ट कठिनाई का चैलेंज 20 ms में हल करता है, और JavaScript अनुक्रमिक प्रूवर 50 गुना धीमा है। उस लैपटॉप पर और चार गुना धीमे फ़ोन पर फ़ॉलबैक हल-समय का अनुमान लगाएँ, और फ़ॉलबैक की रिपोर्ट करने वाले क्लाइंट के लिए एक नीति प्रस्तावित करें (संकेत: वर्कर एक `fallback` संदेश पोस्ट करता है जिसे पेज आगे भेज सकता है)।

अपरिवर्तनीय नियम समझाएँ. समझाएँ कि ब्राउज़र मॉड्यूल को सर्वर के सत्यापनकर्ता स्रोतों से कंपाइल करना दो इम्प्लीमेंटेशन को एक-दूसरे के विरुद्ध परखने से अधिक मज़बूत गारंटी क्यों है।

भाग VIII

प्रायोगिक मूल्यांकन

हम कमिट किए गए हार्नेस द्वारा दर्ज *Sibuna* के मापन प्रस्तुत करते हैं: प्रिमिटिव लेटेंसी, बाहरी लोड जनरेटर के तहत *CPU* और मेमोरी के हिसाब के साथ पूरे उत्पाद, *open-loop* जनरेटर से समान प्रस्तावित लोड पर लेटेंसी, और व्यापक रूप से उपयोग होने वाले *proof-of-work* गेट *Anubis* से तुलना। आठ लाइव डैशबोर्ड वाला एक कंसोल रन रिकॉर्ड को पूरा करता है। इस भाग की हर संख्या किसी कमिट की गई परिणाम फ़ाइल से आती है, जिसका नाम हर तालिका उसके बनने के समय के साथ देती है, या किसी डिज़ाइन रिकॉर्ड से। कार्यात्मक रूप से पास होना और प्रदर्शन का मापन अलग-अलग प्रश्नों के उत्तर देते हैं, और एक होस्ट पर किया गया मापन दूसरे होस्ट पर रैंकिंग नहीं है।

कार्यप्रणाली

उद्देश्य इस अध्याय के अंत तक हमें यह कर पाना चाहिए: इस भाग की हर संख्या दोबारा उत्पन्न करना, बताना कि हर हार्नेस में क्या शामिल है और क्या नहीं, फैलाव वाले कॉलम का अर्थ समझाना, और मापे गए आँकड़े को दर्ज या प्रकाशित आँकड़े से अलग पहचानना।

हार्नेस एक Rust टूल `tools/bench` (बाइनरी `sibuna-bench`) की कमांड हैं, जो `cargo xtask bench` के ज़रिए चलती हैं। एक और टूल कंसोल रन दर्ज करता है।

तालिका 11 हार्नेस और उनकी परिणाम फ़ाइलें, सभी `benchmarks/results/` के अंतर्गत

हार्नेस	यह क्या मापता है	परिणाम फ़ाइल
`cargo xtask bench primitives`	एक CPU पर सोलह प्रिमिटिव वर्कलोड, साथ में हैश किए गए रेट लिमिटर पाथ की दो पंक्तियाँ: सात बैच, min-max फैलाव के साथ प्रति ऑपरेशन माध्यिका	`primitives-latest.json`, `primitives-latest.md`
`cargo xtask bench product`	`wrk` के तहत पूरे डेमन: forward-auth और रिवर्स-प्रॉक्सी मोड में Gate और Shield के लिए थ्रूपुट, लेटेंसी पर्सेंटाइल, प्रति अनुरोध CPU समय और पीक resident मेमोरी	`product-latest.json`, `product-latest.md`
`cargo xtask bench latency`	open-loop जनरेटर से, निश्चित प्रस्तावित दरों पर स्वीकृत अनुरोधों के लेटेंसी पर्सेंटाइल	`latency-latest.json`, `latency-latest.md`
`cargo xtask bench product --anubis`	एक और उत्पाद के रूप में Anubis के साथ पूरे उत्पाद के वर्कलोड, `caddy respond` के पीछे	`comparison-latest.json`, `comparison-latest.md`
`cargo xtask bench admission --anubis`	दो keep-alive क्लाइंट से, forward-auth मोड में Sibuna और Anubis के प्रति सेकंड प्रवेश ऑपरेशन	`admission-latest.json`, `admission-latest.md`

हार्नेस	यह क्या मापता है	परिणाम फ़ाइल
`sibuna-browser dashboards` (`tools/browser`)	एक कार्यात्मक कंसोल रन: headless Chromium में आठ लाइव डैशबोर्ड, जबकि एक जनरेटर ट्रैफ़िक भेजता है	`console-8-dashboards-2026-10-15.md`

`.md` फ़ाइलें सारांश हैं। `.json` फ़ाइलें हर दोहराव, मशीन, revision, और मापी गई हर बाइनरी का पाथ, आकार और SHA-256 रखती हैं। Git इतिहास रखता है, इसलिए डायरेक्टरी केवल सबसे नई फ़ाइलें रखती है। कुछ फ़ाइलें हार्नेस के पुराने संस्करण ने दर्ज की थीं और उनमें उसी रन में मापे गए एक दूसरे डेमन बिल्ड की पंक्तियाँ भी हैं; यह भाग Sibuna की पंक्तियाँ रिपोर्ट करता है।

प्रिमिटिव

हर वर्कलोड अधिकतम 1,000 ऑपरेशन के बिना समय-माप वाले warm-up के बाद सात समय-मापित बैच में चलता है। परिवर्तनशील अवस्था (spent set, रेट लिमिटर) हर बैच से पहले, टाइमर के बाहर, दोबारा बनाई जाती है। टाइमर पूरे बैच को घेरते हैं, कभी अकेले ऑपरेशन को नहीं: प्रति ऑपरेशन घड़ी पढ़ने की लागत उतनी ही होती जितना 100 ns से कम का वह कार्य जिसे वह मापता है। इसलिए प्रति-ऑपरेशन पर्सेंटाइल रिपोर्ट नहीं किए जाते, और फैलाव बैचों के बीच का है। हम प्रति ऑपरेशन माध्यिका बैच रिपोर्ट करते हैं, उसके साथ सबसे तेज़ और सबसे धीमा बैच।

सूट एक CPU पर पिन की गई एक चाइल्ड प्रोसेस (`sibuna-bench primitives-run`) में चलता है। साझा होस्ट पर व्यस्त SMT sibling या कम क्लॉक पूरे सूट रन को धीमा कर देता है, इसलिए टूल सूट को तीन बार (`--rounds`) चलाता है और हर पंक्ति के लिए सबसे कम माध्यिका वाला राउंड रखता है। एक राउंड के भीतर आँकड़े ऊपर जैसे ही हैं।

पूरे उत्पाद

हर मोड (forward auth, रिवर्स प्रॉक्सी) और हर सतह (Gate, Shield) के लिए हार्नेस डेमन को मेमोरी में `--algorithm hashcash --difficulty 8 --rate-limit 100000000 --idle-timeout 5` और चार वर्कर के साथ शुरू करता है, उसके समुच्चय के प्रति CPU एक। यह `sibuna-pow` से डेमन का Hashcash चैलेंज हल करके एक सत्र प्राप्त करता है, जैसे कोई ब्राउज़र करता। फिर यह keep-alive कनेक्शनों पर `wrk` से चार वर्कलोड चलाता है: 2 थ्रेड, 64 कनेक्शन, 5 सेकंड, 3 दोहराव।

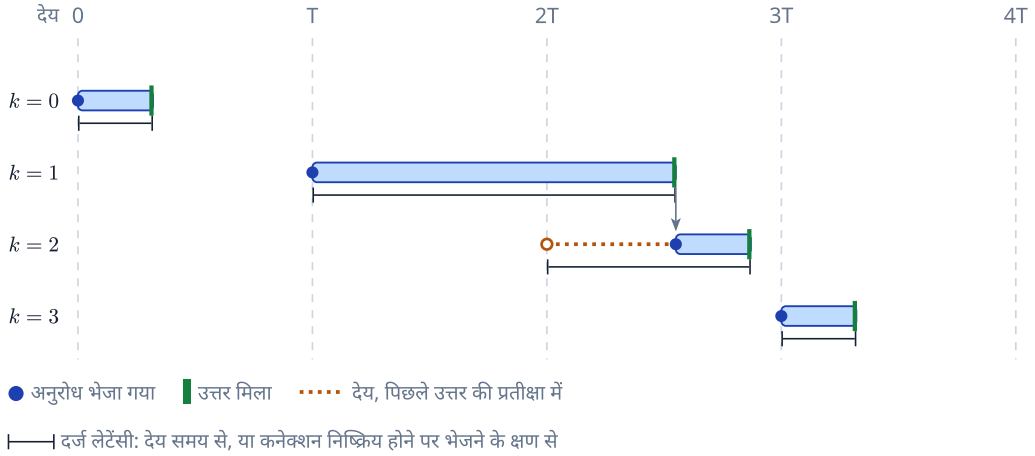
- `admitted`: वैध सत्र कुकी के साथ `GET /private`। रिवर्स-प्रॉक्सी मोड में अनुरोध ऑरिजिन तक पहुँचता है और उसका उत्तर रिले होता है; forward-auth मोड में डेमन स्वयं `200` उत्तर देता है।
- `challenged`: वही अनुरोध बिना कुकी के, ब्राउज़र User-Agent से। डेमन अपना चैलेंज देता है: रिवर्स-प्रॉक्सी मोड में HTML के साथ `200`, forward-auth मोड में `401`।
- `allowed_static`: `curl/8.0` से `GET /robots.txt`, जिसे दोनों सतहें बिना चैलेंज के प्रवेश देती हैं।
- `attack`: वैध सत्र और साथ में `q=' OR 1=1--`। केवल Shield इसे अस्वीकार करता है (`403`)। Gate इसे प्रवेश देता है, और रिवर्स-प्रॉक्सी मोड में इसे ऑरिजिन तक रिले करता है।

ऑरिजिन एक स्टब (`sibuna-bench origin`) है: केवल standard library, प्रति कनेक्शन एक थ्रेड, keep-alive, और 41-बाइट का पेज। प्रति अनुरोध CPU समय पूरे रन में `/proc/<pid>/stat` से डेमन का user और system समय (10 ms tick) है, जिसे `wrk` द्वारा गिने गए अनुरोधों से भाग दिया जाता है; ऑरिजिन का CPU शामिल नहीं है। पीक मेमोरी `/proc/<pid>/status` से `VmHWM` है, जिसे हर रन से पहले `/proc/<pid>/clear_refs` के ज़रिए reset किया जाता है, ताकि हर रन अपना पीक रिपोर्ट करे। हर आँकड़ा दोहरावों की माध्यिका है, और न्यूनतम तथा अधिकतम उसके साथ JSON फ़ाइल में रखे जाते हैं।

`wrk` एक closed-loop जनरेटर है: यह 64 अनुरोध एक साथ चलते रखता है। इसलिए तेज़ डेमन को अधिक अनुरोध मिलते हैं, और `wrk` के पर्सेंटाइल उन अनुरोधों को छोड़ देते हैं जो उसने धीमे उत्तर की प्रतीक्षा करते समय नहीं भेजे (coordinated omission)। इसका p99 दिखाता है कि डेमन अपनी संतृप्ति में कैसा व्यवहार करता है, किसी दिए गए लोड पर उसकी लेटेंसी नहीं। लेटेंसी हार्नेस उस प्रश्न का उत्तर देता है।

समान लोड पर लेटेंसी

open-loop जनरेटर (`sibuna-bench generate`) प्रति कनेक्शन एक blocking थ्रेड चलाता है। 64 कनेक्शनों में से हर एक एक निश्चित समय-सारणी पर सत्र के साथ `GET /private` भेजता है: किसी कनेक्शन का अनुरोध k समय $t_0 + o + kT$ पर देय है, जहाँ o कनेक्शन का offset है और T उसका अंतराल, पहले के उत्तरों से स्वतंत्र। जब कोई उत्तर अगले अनुरोध के देय होने के बाद आता है, तो वह अनुरोध तुरंत भेजा जाता है और उसकी लेटेंसी उसके देय समय से गिनी जाती है, जैसे `wrk2` में। जब देय समय पर कनेक्शन निष्क्रिय था, तो लेटेंसी वास्तविक भेजने के समय से गिनी जाती है, ताकि जनरेटर की अपनी sleep की देरी डेमन के खाते में न जाए।

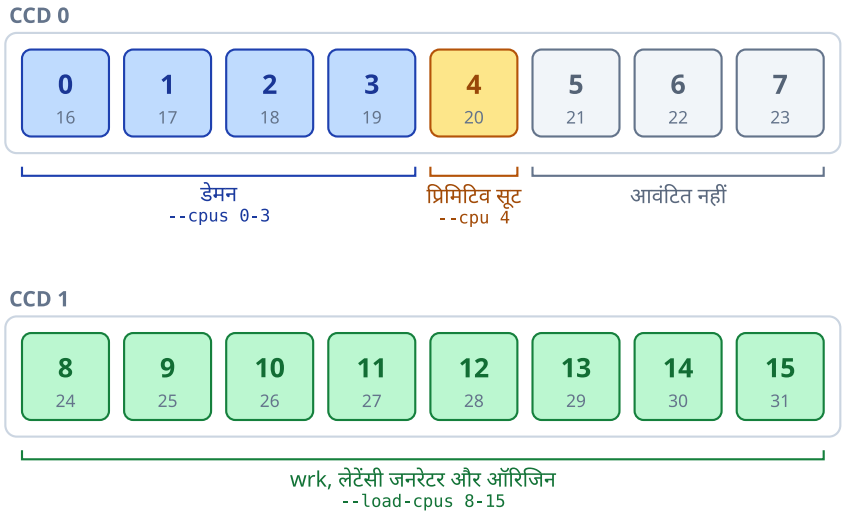


चित्र 10 open-loop जनरेटर का एक कनेक्शन। अनुरोध 1 का उत्तर अनुरोध 2 के देय होने के बाद आता है, इसलिए अनुरोध 2 तुरंत भेजा जाता है और उसकी लेटेंसी 2T से गिनी जाती है; अनुरोध 3 कनेक्शन को निष्क्रिय पाता है और उसकी लेटेंसी उसके भेजे जाने से गिनी जाती है।

हार्नेस Shield सतह और स्वीकृत वर्कलोड को पाँच प्रस्तावित दरों पर चलाता है: प्रति सेकंड 10,000, 25,000, 50,000, 100,000 और 150,000 अनुरोध, हर एक 2-सेकंड warm-up, 10 मापे गए सेकंड और 3 रन के साथ। हार्नेस एक रन में दो डेमन बिल्ड माप सकता है: तब वह दोनों को एक ही CPU समुच्चय पर शुरू करता है और हर दर और दोहराव पर उनके बीच बारी-बारी करता है, निष्क्रिय वाला कोई CPU उपयोग नहीं करता, ताकि मशीन का धीमा विचलन दोनों को समान रूप से प्रभावित करे। दर्ज रन ने ऐसा ही किया। पर्सेंटाइल एक रन के हर मापे गए अनुरोध पर nearest-rank हैं; हर आँकड़ा तीन रन की माध्यिका है, और उसके साथ p99 का min-max।

CPU समुच्चय

CPU समुच्चय किसी भी अन्य सेटिंग से अधिक मायने रखते हैं। डेमन CPU 0 से 3 पर चलते हैं (`--cpus`; `--workers` उस समुच्चय का आकार है)। `wrk`, लेटेंसी जनरेटर और ऑरिजिन CPU 8 से 15 पर चलते हैं (`--load-cpus`)। प्रिमिटिव सूट CPU 4 पर चलता है (`--cpu`)। हर प्रोसेस शुरू होने से पहले `taskset -c` से पिन की जाती है, इसलिए वह जो भी थ्रेड बनाती है वह समुच्चय विरासत में पाता है। प्रवेश हार्नेस उत्पाद के लिए दो CPU, 0 और 1, और अपने क्लाइंट के लिए CPU 8 से 15 उपयोग करता है।



चित्र 11 AMD Ryzen 9 5950X पर CPU समुच्चय। CPU 0 से 7 पहले CCD के आठ कोर हैं, 8 से 15 दूसरे के, और 16 से 31 उनके SMT sibling। डेमन और लोड जनरेटर कभी कोई कोर साझा नहीं करते।

दूसरी मशीनों पर हम ऐसे समुच्चय चुनते हैं जो कोर साझा न करें (`lscpu -e` टोपोलॉजी दिखाता है)।

मशीन और परिस्थितियाँ

फ़ाइलें `paxos-zig` पर दर्ज की गईं: एक AMD Ryzen 9 5950X 16-Core Processor (32 लॉजिकल CPU), Linux 7.0.0-28-generic x86_64, boost सक्षम के साथ `powersave` governor, rustc 1.99.0। प्रिमिटिव, उत्पाद और लेटेंसी फ़ाइलें 15 अक्टूबर 2026 को दर्ज की गईं, और Anubis के साथ तुलना 16 अक्टूबर 2026 को। डेमन `cargo build --release -p sibuna` है, SQLite स्टोरेज और कंसोल के साथ कंपाइल हुआ डिफ़ॉल्ट बिल्ड (SID 0010); उत्पाद और लेटेंसी फ़ाइलें 49,113,856 बाइट की बाइनरी का नाम देती हैं। हर फ़ाइल अपना revision (प्रिमिटिव के लिए `efce14d`, उत्पाद और लेटेंसी रन के लिए `249d913`) `dirty: true` के साथ दर्ज करती है; दूसरे वाले के working tree में पहले से वे दोबारा उपयोग होने वाले कनेक्शन थ्रेड थे जिन्हें SID 0002 दर्ज करता है, और SID 0002 उनके प्रभाव के लिए `latency-latest.md` का हवाला देता है। तुलना और प्रवेश फ़ाइलें `dirty: true` के साथ revision `54e4ec5` का नाम देती हैं, क्योंकि working tree में `tools/bench` में हार्नेस के जोड़ थे; उनका डेमन उसी revision का रिलीज़ बिल्ड है (37,450,232 बाइट)। उनके एक-मिनट load average तुलना रन से पहले 3.53 और प्रवेश रन समाप्त होने पर 0.56 थे।

फ़ाइलें प्रिमिटिव रन पर 2.46, लेटेंसी रन से पहले 1.30 और बाद में 52.31, और उत्पाद रन से पहले 52.31 और बाद में 33.39 के एक-मिनट load average दर्ज करती हैं। ऊँचे आँकड़े लेटेंसी जनरेटर के 64 थ्रेड हैं, जिसके तुरंत बाद उत्पाद रन चला।

> **मापन का दायरा** केवल स्थानीय मापन दिए जाते हैं। हर रन loopback पर एक होस्ट का उपयोग करता है: कोई TLS नहीं, कोई WAN नहीं, कोई वास्तविक ब्राउज़र नहीं। डेमन स्टोरेज के बिना मेमोरी में चलते हैं। टाइमर बैच को घेरते हैं; अवस्था reset और warm-up बिना समय-माप के होते हैं। हार्नेस आवंटन नहीं गिनता (फ़ाइलें कहती हैं "not instrumented; API/source audit only"); आवंटन-मुक्त अनुरोध पाथ की जाँच इसके बजाय भाग V

> के steady-state परीक्षण करते हैं। कोई हार्नेस डेमन में लिंक नहीं होता या अनुरोध पाथ में hook नहीं जोड़ता: उत्पाद और लेटेंसी हार्नेस अपरिवर्तित रिलीज़ बाइनरी चलाते हैं, जिसकी पहचान उसका SHA-256 है। इस होस्ट पर रैंकिंग किसी दूसरे पर रैंकिंग नहीं है।

दोबारा कैसे उत्पन्न करें

हमें Linux, `taskset` (util-linux) और `PATH` पर `wrk` चाहिए।

```
cargo xtask bench primitives
cargo xtask bench product
cargo xtask bench latency
cargo xtask bench help
```

`cargo xtask bench` पहले `cargo build --release -p sibuna` से डेमन बनाता है, फिर `sibuna-bench` को रिलीज़ मोड में चलाता है। टूल डेमन को अपनी बाइनरी के पास पाता है; `--rust <path>` इसे बदलता है। `cargo xtask bench help` विकल्पों की सूची देता है, जिनमें `--cpus`, `--load-cpus`, `--cpu`, `--rounds`, `--repetitions`, `--seconds`, `--rates` और `--warmup` हैं। नीचे Anubis वाला अनुभाग वे कमांड देता है जिनसे वह तुलना दर्ज हुई।

प्रिमिटिव लेटेंसी

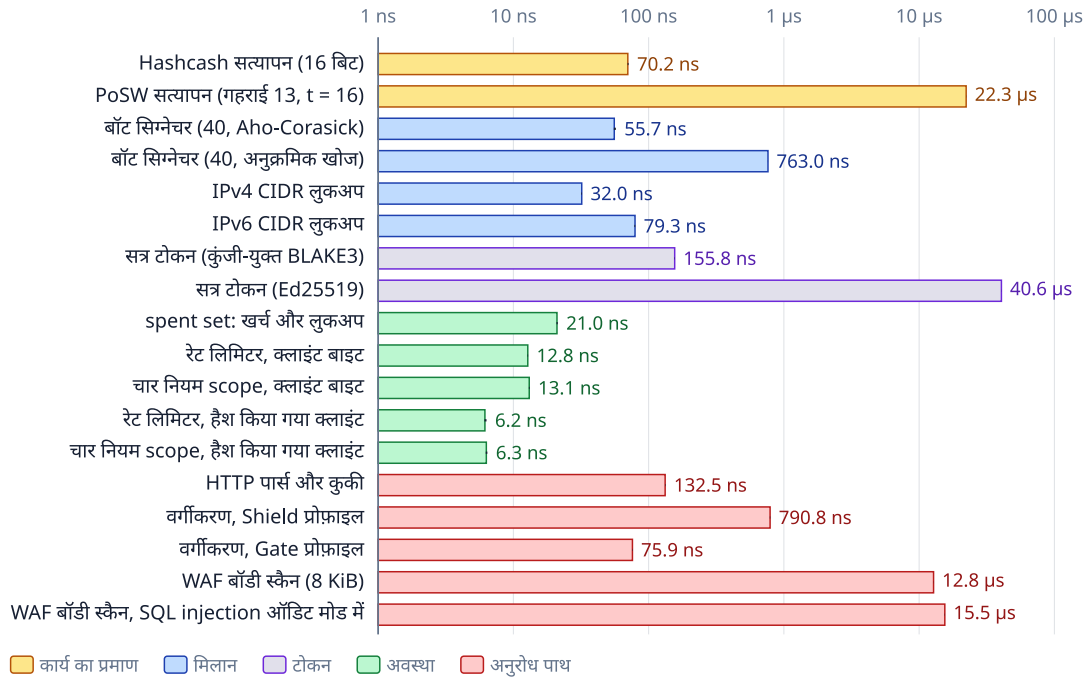
तालिका प्रिमिटिव रन की हर पंक्ति देती है। JSON फ़ाइल हर पंक्ति को सबसिस्टम और वर्कलोड से नाम देती है (उदाहरण के लिए `pow_verify` और `hashcash_16_bits`)।

तालिका 12 एक Ryzen 9 5950X कोर पर प्रति ऑपरेशन नैनोसेकंड में प्रिमिटिव समय-माप (`benchmarks/results/primitives-latest.md`, 2026-10-15T22:34:59Z पर बनी)

वर्कलोड	ns/op	min-max
Hashcash सत्यापन (16 बिट)	70.2	69.9-71.0
PoSW सत्यापन (गहराई 13, t = 16)	22272.4	22257.0-22316.4
बॉट सिग्नेचर (40, Aho-Corasick)	55.7	55.3-57.2
बॉट सिग्नेचर (40, अनुक्रमिक खोज)	763.0	762.2-763.5
IPv4 CIDR लुकअप	32.0	31.8-32.2
IPv6 CIDR लुकअप	79.3	78.5-79.6
सत्र टोकन (कुंजी-युक्त BLAKE3)	155.8	155.3-156.4
सत्र टोकन (Ed25519)	40587.5	40544.6-40631.1
spent set (Robin Hood), खर्च और लुकअप	21.0	20.8-21.3
रेट लिमिटर (GCRA), क्लाइंट बाइट	12.8	12.7-12.8
रेट लिमिटर, चार नियम scope, क्लाइंट बाइट	13.1	13.0-13.2
रेट लिमिटर (GCRA), हैश किया गया क्लाइंट	6.2	6.1-6.3
रेट लिमिटर, चार नियम scope, हैश किया गया क्लाइंट	6.3	6.2-6.4

वर्कलोड	ns/op	min-max
HTTP पार्स और कुकी	132.5	132.3-133.2
वर्गीकरण, Shield प्रोफ़ाइल	790.8	787.2-793.1
वर्गीकरण, Gate प्रोफ़ाइल	75.9	75.6-76.1
WAF बॉडी स्कैन (8 KiB)	12784.3	12771.5-12809.2
WAF बॉडी स्कैन (8 KiB), SQL injection ऑडिट मोड में	15485.4	15460.9-15530.1

फैलाव संकरे हैं: किसी भी पंक्ति का कोई बैच अपनी पंक्ति की माध्यिका से 2.8% से अधिक दूर नहीं है। चित्र यही पंक्तियाँ लघुगणकीय पैमाने पर दिखाता है, जिससे 6 ns और 40 us की पंक्तियाँ साथ-साथ पढ़ने योग्य रहती हैं।



हर पट्टी AMD Ryzen 9 5950X 16-Core Processor के एक कोर पर 7 बैच की माध्यिका है, 2026-10-15; पतली रेखा सबसे तेज़ से सबसे धीमे बैच तक फैली है। लघुगणकीय पैमाना।

चित्र 12 लघुगणकीय पैमाने पर प्रिमिटिव के समय, `benchmarks/results/primitives-latest.json` से बनाए गए।

Proof-of-work सत्यापन

Hashcash सत्यापन एक चैलेंज, एक separator और एक दशमलव नॉनस का हैश करता है: 70.2 ns। अनुक्रमिक कार्य का सत्यापनकर्ता ग्राफ़ की ओपनिंग जाँचता है: 22,272.4 ns। इसके हैश कॉल और SHA-256 compression की गिनती Hashcash से अलग है, क्योंकि इनपुट की लंबाई अलग है। दोनों पंक्तियाँ SHA-256 से बँधी हैं, और `sibuna-crypto` CPU में उपलब्ध होने पर SHA-NI निर्देशों का उपयोग करता है, जिनका पता रन टाइम पर लगता है (SID 0010)।

मिलान

सघन Aho-Corasick ऑटोमेटन और अनुक्रमिक सबस्ट्रिंग खोज एक जैसे पैटर्न और any-match अर्थ (semantics) का उपयोग करते हैं। उन्हीं 40 सिग्नेचर पर ऑटोमेटन अनुक्रमिक खोज से 13.7 गुना तेज़ है (55.7 बनाम 763.0 ns)। इसकी टेबल प्रति अवस्था 256 कॉलम के बजाय byte class का उपयोग करती हैं (SID 0006, Part F), जिससे बॉट सिग्नेचर टेबल 42,244 बाइट पर रहती है (नीचे की footprint तालिका)। CIDR वर्गीकारक 32.0 ns (IPv4) और 79.3 ns (IPv6) लेते हैं।

टोकन

BLAKE3 MAC टोकन का सत्यापन 155.8 ns में होता है। `sibuna-crypto` BLAKE3 को पोर्टेबल रूप में, बिना SIMD पाथ के, लागू करता है; SIMD पाथ बड़े इनपुट पर जीतते हैं, लेकिन Sibuna BLAKE3 से 100 बाइट से छोटे इनपुट का हैश करता है, जहाँ पोर्टेबल कोड तेज़ है। Ed25519 compact टोकन का सत्यापन 40,587.5 ns में होता है: सत्यापनकर्ता width-5 और width-8 NAF अंकों का उपयोग करता है और डिकोड की गई सार्वजनिक कुंजी रखता है। एक Ed25519 सत्यापन की लागत लगभग 260 MAC सत्यापनों के बराबर है; MAC Sibuna की डिफ़ॉल्ट टोकन योजना है।

अवस्था

spent-set पंक्ति एक नया टैग डालती है और उसे फिर से खोजती है: 21.0 ns। इस पंक्ति की टेबल में 4,096 स्लॉट के 16 शार्ड हैं (65,536 टैग; डेमन का डिफ़ॉल्ट 1,048,576 है), और इसे हर बैच से पहले, टाइमर के बाहर, उसी स्थान पर खाली किया जाता है। इसलिए हर बैच resident मेमोरी को probe करता है। सूट का एक पुराना संस्करण हर बैच के लिए डेमन के डिफ़ॉल्ट आकार की एक नई शून्य की गई टेबल बनाता था; वह पहले राइट पर टेबल के पेज को कर्नेल द्वारा मैप करने का समय मापता था, एक ऐसी लागत जो चलता हुआ डेमन एक बार चुकाता है, और वह पंक्ति हम रिपोर्ट नहीं करते। रेट लिमिटर की पंक्तियों को सावधानी चाहिए। क्लाइंट बाइट वाली दो पंक्तियाँ (`gcra_check` और `gcra_four_rule_scopes`) byte API को कॉल करती हैं: हर जाँच क्लाइंट का पता बाइट के रूप में लेती है और पहले ऑपरेटिंग सिस्टम की यादृच्छिकता से लिए गए एक सीक्रेट के तहत उसका कुंजी-युक्त SipHash-1-3 निकालती है। सीक्रेट कुंजी किसी को भी ऐसे पते निकालने से रोकती है जो किसी पीड़ित की probe विंडो साझा करें (SID 0010 और `sibuna-store` का दस्तावेज़ीकरण)। ये पंक्तियाँ 12.8 और 13.1 ns लेती हैं।

डेमन अनुरोध पाथ पर byte API का उपयोग नहीं करता। यह प्रति अनुरोध क्लाइंट का एक बार हैश करता है (`ClientHasher`) और वह हैश हर उस टेबल को देता है जिससे वह परामर्श करता है। हैश किए गए क्लाइंट वाली दो पंक्तियाँ (`gcra_check_hashed` और `gcra_four_rule_scopes_hashed`) उसी पाथ को मापती हैं, हैश लूप से पहले निकालकर: प्रति जाँच 6.2 और 6.3 ns। byte और hashed पंक्तियों की माध्यिकाओं के बीच का अंतर, 6.6 ns, इन इनपुट पर कुंजी-युक्त हैश की लगभग कीमत है, जिसे डेमन प्रति टेबल नहीं, प्रति अनुरोध एक बार चुकाता है।

HTTP पार्सर

पार्सर पंक्ति 400-बाइट के ब्राउज़र अनुरोध को पार्स करती है और एक कुकी खोजती है: 132.5 ns। क्रेट का अपना समय-परीक्षण इससे सहमत है (`crates/http`: उसी मशीन पर लगभग 135 से 180 ns)। पार्सर zero-copy है; हर फ़ील्ड कनेक्शन के बफ़र की एक स्लाइस है।

नीति इंजन

ब्राउज़र अनुरोध का पूरा वर्गीकरण 790.8 ns लेता है और Gate प्रोफ़ाइल 75.9 ns। ये पंक्तियाँ अनुरोध के पार्स किए गए क्लाइंट पते को खाली छोड़ती हैं, इसलिए इंजन मूल्यांकन के भीतर क्लाइंट पते के टेक्स्ट को पार्स करता है। डेमन पते को प्रति अनुरोध एक बार, मूल्यांकन से पहले, पार्स करता है, इसलिए इन पंक्तियों में एक छोटी लागत शामिल है जिसे डेमन इंजन के बाहर चुकाता है। इंजन की हीप टेबल कुल 209,052 बाइट हैं (SID 0002)।

WAF निरीक्षण

8 KiB बॉडी का सिमैंटिक स्कैन 12,784.3 ns लेता है, यानी 12784.3/8192 ≈ 1.56 ns प्रति बाइट। दूसरी पंक्ति उसी बॉडी को SQL injection श्रेणी ऑडिट मोड में और बाकी enforce मोड में रखकर स्कैन करती है: 15,485.4 ns। जब कोई श्रेणी enforce के बजाय ऑडिट मोड में हो, तो हर फ़ील्ड का निरीक्षण प्रति श्रेणी होता है (SID 0004, "inspection modes")। इंजन फ़ील्ड को सिग्नेचर ऑटोमेटन से एक बार स्कैन करता है, उन श्रेणियों के समुच्चय के लिए जिनके सिग्नेचर आते हैं, फिर उनमें से केवल वांछित श्रेणियाँ खोजता है, इसलिए एक श्रेणी को ऑडिट करने की लागत सभी को enforce करने से लगभग पाँचवाँ हिस्सा अधिक है, प्रति श्रेणी एक स्कैन नहीं।

पूरे उत्पादों की तुलना

> **उद्देश्य** एक पूरी प्रोसेस के रूप में डेमन का थ्रूपुट, लेटेंसी, CPU समय और मेमोरी पढ़ें, उसकी सतहों और मोड की तुलना करें, और बताएँ कि किन पंक्तियों में सीमा डेमन तय करता है और किनमें लोड जनरेटर।

तालिकाएँ हर सतह और वर्कलोड के लिए पाँच-सेकंड के तीन `wrk` रन की माध्यिकाएँ देती हैं। हर वर्कलोड के पास का स्टेटस वह उत्तर है जिसकी हार्नेस ने अपेक्षा की और जो उसे मिला।

Forward-auth मोड

इनग्रेस डेमन से पूछता है कि किसी अनुरोध को प्रवेश दे या नहीं; कोई ऑरिजिन नहीं और कोई बॉडी रिले नहीं, इसलिए यह प्रवेश की लागत का सबसे शुद्ध माप है।

तालिका 13 Forward-auth मोड, Gate और Shield; लेटेंसी माइक्रोसेकंड में (`benchmarks/results/product-latest.md`, 2026-10-15T23:15:24Z पर बनी)

सतह	वर्कलोड (स्टेटस)	req/s	p50	p90	p99	CPU us/req	पीक RSS KiB
Gate	`admitted` (200)	211,345	151	175	305	11.84	15912
Gate	`challenged` (401)	198,315	160	180	326	13.15	16176
Gate	`allowed_static` (200)	211,053	152	175	305	11.73	16176
Gate	`attack` (200)	211,618	151	176	305	11.84	16176
Shield	`admitted` (200)	208,101	152	178	308	12.87	15928
Shield	`challenged` (401)	197,532	161	183	327	13.77	16192
Shield	`allowed_static` (200)	211,636	151	175	305	11.78	16192
Shield	`attack` (403)	210,549	152	175	307	12.32	16192

forward-auth मोड में डेमन बाधा (bottleneck) नहीं है। उसके व्यस्त कोर (वास्तविक समय पर CPU समय, JSON फ़ाइल में `cores_busy_median`) चार डेमन CPU में से 2.47 और 2.72 के बीच रहते हैं, और हर पंक्ति प्रति सेकंड 197,532 और 211,636 अनुरोधों के बीच है। यह लोड जनरेटर को ऊपरी सीमा के रूप में दिखाता है। उस सीमा के पास पंक्तियों के बीच कुछ प्रतिशत का अंतर रन दर रन बदलता है: अकेली Gate `admitted` पंक्ति अपने तीन रन में प्रति सेकंड 205,151 से 211,513 अनुरोधों तक फैली है। इस मोड में अधिक उपयोगी आँकड़ा प्रति अनुरोध CPU समय है: 11.73

से 13.77 us, Gate की तुलना में Shield के लिए थोड़ा अधिक, और सबसे अधिक `challenged` पंक्तियों के लिए, जो चैलेंज उत्तर रेंडर करती हैं।

रिवर्स-प्रॉक्सी मोड

डेमन स्टब ऑरिजिन के सामने रहता है और स्वीकृत अनुरोधों और रिस्पॉन्स को रिले करता है। इसलिए प्रॉक्सी की गई पंक्तियों में ऑरिजिन की अपनी लागत और एक अतिरिक्त loopback hop शामिल है।

तालिका 14 रिवर्स-प्रॉक्सी मोड, Gate और Shield; लेटेंसी माइक्रोसेकंड में (`benchmarks/results/product-latest.md`, 2026-10-15T23:15:24Z पर बनी)

सतह	वर्कलोड (स्टेटस)	req/s	p50	p90	p99	CPU us/req	पीक RSS KiB
Gate	`admitted` (200)	142,498	427	548	768	27.68	15896
Gate	`challenged` (200)	199,465	160	181	324	12.98	16160
Gate	`allowed_static` (200)	147,303	411	541	759	26.88	16324
Gate	`attack` (200)	143,900	422	555	745	27.67	16348
Shield	`admitted` (200)	138,965	438	571	857	28.57	16064
Shield	`challenged` (200)	195,188	162	185	328	13.73	16328
Shield	`allowed_static` (200)	145,673	415	551	746	27.27	16328
Shield	`attack` (403)	209,507	151	177	306	12.17	16328

यहाँ प्रॉक्सी की गई पंक्तियाँ (`admitted`, `allowed_static`, और Gate का `attack`, जिसे Gate रिले करता है) CPU से बँधी हैं: चार में से 3.96 से 3.98 कोर व्यस्त हैं। वे प्रति अनुरोध 26.88 से 28.57 us CPU पर प्रति सेकंड 138,965 से 147,303 अनुरोध सर्व करती हैं, जो उस उत्तर की लागत का लगभग दोगुना है जिसे डेमन स्वयं देता है। `challenged` पंक्तियों और Shield की `attack` पंक्ति का उत्तर डेमन स्वयं देता है, और वे forward-auth पंक्तियों जैसी दिखती हैं। संतृप्ति में `wrk` का p99 डेमन की अपनी दर पर शेड्यूलर की टेल है, इसलिए हम इसके बजाय अगले अध्याय की टेल पढ़ते हैं।

तीन रिवर्स-प्रॉक्सी पंक्तियों में `wrk` ने 2xx और 3xx वर्गों से बाहर के रिस्पॉन्स गिने, जहाँ किसी की अपेक्षा नहीं थी: 2,162,579 में से 3 अनुरोध (Gate `attack`), 2,082,640 में से 1 (Shield `admitted`) और 2,204,358 में से 1 (Shield `allowed_static`), तीनों रन का योग। फ़ाइलें यह दर्ज नहीं करतीं कि ये कौन-सा स्टेटस थे। डेमन के मेट्रिक के साथ रन को दोहराने पर `sibuna_overloaded_total` में गिने गए `503` उत्तर दिखे: `wrk` अपने 64 कनेक्शन एक पते से खोलता है, जो ठीक डिफ़ॉल्ट `--max-connections-per-ip` है, और जब कोई कनेक्शन अपने 4,096वें अनुरोध के बाद बंद हुआ, तो पुराने कनेक्शन के अपना स्थान लौटाने से पहले ही `wrk` दोबारा जुड़ गया। अब डेमन समापन रिस्पॉन्स लिखने से पहले स्थान लौटा देता है (SID 0002, "bug fix"); तालिकाएँ उस सुधार से पहले की हैं।

संसाधन उपयोग (Footprint)

तालिका 15 डेमन के आकार और resident मेमोरी (`benchmarks/results/product-latest.json`, 2026-10-15T23:15:24Z पर बनी, और `primitives-latest.json`, 2026-10-15T22:34:59Z पर बनी)

माप	मान
डेमन बाइनरी	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
निष्क्रिय resident मेमोरी, प्रिमिटिव रन	6,052 KiB
नीति इंजन टेबल	209,052 बाइट (हीप)
बॉट सिग्नेचर टेबल	42,244 बाइट (हीप)
WAF टेबल	164,908 बाइट (हीप)
ब्राउज़र सॉल्वर मॉड्यूल (WebAssembly)	14,744 bytes

श्रेणियाँ चार उत्पाद रन (दो मोड, दो सतह) पर हैं, और पीक के लिए उनके सोलह वर्कलोड पर। टेबल के आकार इंजन द्वारा रिपोर्ट किए गए हीप आकार हैं। रिकॉर्ड बाइनरी के आकार का विभाजन नहीं देता।

समान प्रस्तावित लोड पर लेटेंसी

> **उद्देश्य** किसी निश्चित प्रस्तावित लोड पर लेटेंसी वितरण पढ़ें, टेल पर्सेंटाइल को उसके फैलाव के साथ पढ़ें, और टेल के एक दोष को उसके मापन से उसके सुधार तक देखें।

दोनों तालिकाएँ `cargo xtask bench latency` के एक रन से आती हैं। त्रुटियों वाला कॉलम जनरेटर की कनेक्शन, I/O और फ़्रेमिंग त्रुटियाँ गिनता है।

तालिका 16 Forward-auth मोड, Shield, निश्चित प्रस्तावित दरों पर स्वीकृत अनुरोध; लेटेंसी माइक्रोसेकंड में (`benchmarks/results/latency-latest.md`, 2026-10-15T23:07:14Z पर बनी)

प्रस्तावित req/s	प्राप्त req/s	p50	p90	p99 (min-max)	p99.9	अधिकतम	त्रुटियाँ	CPU us/req
10,000	10,000	30	32	37 (36-39)	58	355	0	16.92
25,000	25,000	28	31	34 (32-34)	45	1035	0	15.83
50,000	50,000	25	27	30 (30-30)	40	3029	0	13.83
100,000	99,998	24	26	41 (40-42)	62	2358	0	13.48
150,000	149,996	25	36	51 (51-52)	74	1753	0	14.16

तालिका 17 रिवर्स-प्रॉक्सी मोड, Shield, निश्चित प्रस्तावित दरों पर स्वीकृत अनुरोध; लेटेंसी माइक्रोसेकंड में (`benchmarks/results/latency-latest.md`, 2026-10-15T23:07:14Z पर बनी)

प्रस्तावित req/s	प्राप्त req/s	p50	p90	p99 (min-max)	p99.9	अधिक-तम	त्रुटियाँ	CPU us/ req
10,000	10,000	56	61	65 (65-66)	76	555	0	32.25
25,000	25,000	51	54	58 (58-60)	72	1105	0	28.63
50,000	50,000	47	50	60 (58-66)	74	3066	0	27.45
100,000	99,996	55	75	100 (100-104)	150	2239	0	28.42
150,000	134,344	583934	154416	01906273 (1631234-2218760)	1967134	1989964	4	29.22

forward-auth मोड में p50 हर दर पर 24 और 30 us के बीच रहता है। p99 प्रति सेकंड 150,000 अनुरोधों तक 51 us या उससे कम रहता है, और p99.9 74 us या उससे कम। डेमन प्रति अनुरोध 13.48 से 16.92 us CPU खर्च करता है। रिवर्स-प्रॉक्सी मोड में p50 प्रति सेकंड 100,000 अनुरोधों तक 47 से 56 us है, और 100,000 पर p99 100 us तथा p99.9 150 us है। प्रति सेकंड 150,000 अनुरोधों पर डेमन संतृप्त हो जाता है: वह प्रति सेकंड 134,344 अनुरोध पूरे करता है, और लेटेंसी सेकंडों तक बढ़ जाती है, जैसा कि क्षमता से ऊपर open-loop परीक्षण को दिखाना चाहिए। वह दर पिछले अध्याय के closed-loop प्रॉक्सी थ्रूपुट के पास है (Shield `admitted` के लिए प्रति सेकंड 138,965 अनुरोध)। उस पंक्ति की चार त्रुटियाँ ओवरलोड में हुईं।

संतृप्ति से नीचे p99 के फैलाव संकरे हैं; सबसे चौड़ा प्रति सेकंड 50,000 अनुरोधों पर रिवर्स-प्रॉक्सी p99 है, 58 से 66 us। अधिकतम वाला कॉलम एक रन के 15 लाख तक के अनुरोधों में से एक अनुरोध है। साझा होस्ट पर हम इसे डेमन का गुण नहीं, होस्ट का सबसे खराब क्षण मानते हैं।

मापन ने क्या बदला

बाहरी लोड ने Sibuna का डिज़ाइन बदला है। डेमन हर कनेक्शन को उसके अपने थ्रेड पर सर्व करता है (SID 0002, SID 0010), और उसके पहले लेटेंसी रन ने एक ऐसी टेल पाई जो `wrk` ने नहीं दिखाई थी। 64 कनेक्शन के साथ, जिनमें से हर एक को डेमन 4,096 अनुरोधों के बाद बंद करता है, प्रति सेकंड 100,000 स्वीकृत अनुरोधों पर 99.9वाँ पर्सेंटाइल 0.3 से 2 ms था।

SID 0002 में और `cmd/sibuna/src/tasks.rs` के दस्तावेज़ीकरण में दर्ज निदान थ्रेड का बदलाव (turnover) है। 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 डेमन के दूसरे थ्रेड को अटका दिया। एक प्रयोग ने इसे तय कर दिया: एक बिल्ड जो कभी कनेक्शन बंद नहीं करता था, ताकि रन के दौरान कोई थ्रेड न बने, उसने उसी दर पर 54 से 60 us दिए। सुधार कनेक्शन थ्रेड को दोबारा उपयोग करता है: जो थ्रेड अपना कनेक्शन पूरा कर लेता है, वह रुककर अगला ले लेता है, इसलिए नए कनेक्शनों की स्थिर धारा कोई थ्रेड नहीं बनाती। 10 सेकंड तक रुका रहा थ्रेड समाप्त हो जाता है और अगले launch या task पूरा करने वाले अगले थ्रेड द्वारा join किया जाता है, इसलिए कोई बर्स्ट अपने स्टैक मेमोरी में नहीं छोड़ता। जिस थ्रेड का कनेक्शन panic करता है, वह रुकने के बजाय समाप्त हो जाता है। पहले की तरह एक साथ अधिकतम `--max-connections` कनेक्शन सर्व होते हैं।

ऊपर की तालिकाएँ सुधार के बाद के रन की हैं: प्रति सेकंड 100,000 अनुरोधों पर forward-auth मोड में 62 us और रिवर्स-प्रॉक्सी मोड में 150 us का p99.9। सुधार से पहले के रन परिणाम फ़ाइलों के रूप में नहीं रखे गए, इसीलिए उनके आँकड़े केवल SID 0002 में और इस अनुभाग में दिखते हैं।

हल किया गया उदाहरण: रुकावट p99.9 पर क्यों दिखी 64 कनेक्शन पर प्रति सेकंड 100,000 अनुरोधों पर हर कनेक्शन प्रति सेकंड $100000/64 = 1562.5$ अनुरोध ले जाता है। डेमन keep-alive कनेक्शन को 4,096 अनुरोधों के बाद बंद करता है (`cmd/sibuna/src/connection.rs` में `MAX_REQUESTS_PER_CONNECTION`), इसलिए एक कनेक्शन $4096/1562.5 \approx 2.62$ s चलता है, और जनरेटर प्रति सेकंड लगभग $64/2.62 \approx 24.4$ नए कनेक्शन खोलता है। प्रति कनेक्शन एक थ्रेड के साथ, यह हर सेकंड लगभग 24 थ्रेड शुरू होना और 24 थ्रेड समाप्त होना है। इस दर पर 10-सेकंड का रन 1,000,000 अनुरोध मापता है। इसके p99.9 से उनमें से 1,000 आगे होते हैं, प्रति सेकंड 100; इसके p99 से प्रति सेकंड 1,000। यदि हर turnover दूसरे थ्रेड पर चल रहे लगभग $100/24.4 \approx 4$ अनुरोधों को मिलीसेकंड के एक अंश से विलंबित करता है, तो केवल विलंबित अनुरोध ही ऊपरी 0.1% को भर देते हैं, जबकि ऊपरी 1% अब भी सामान्य कतारबद्धता से बना होता है। एक दुर्लभ, आवर्ती रुकावट p99 को हिलाने से बहुत पहले दूर की टेल को हिला देती है। इसीलिए ऐसे दोष को देखने के लिए p99.9 और हर अनुरोध पर टाइमर चाहिए।

Sibuna और Anubis

उद्देश्य समान लोड के तहत एक स्थापित उत्पाद के आँकड़ों के साथ Sibuna के आँकड़े पढ़ें, और बताएँ कि कौन-से अंतर उत्पादों से आते हैं और कौन-से सेटअप से।

Techaro का Anubis एक व्यापक रूप से उपयोग होने वाला proof-of-work गेट है जो वेबसाइटों के सामने रहता है, लगभग उसी तरह जैसे Sibuna का Gate। हमने Anubis v1.27.0 की आधिकारिक रिलीज़ बाइनरी (`anubis-1.27.0-linux-amd64`, 40,001,698 बाइट) को उसी होस्ट पर, उन्हीं CPU पर, उन्हीं वर्कलोड के साथ Sibuna के साथ मापा। दोनों परिणाम फ़ाइलें इनसे दर्ज की गईं:

```
cargo xtask bench product --anubis <path/to/anubis>
cargo xtask bench admission --anubis <path/to/anubis>
```

Anubis को वैसे ही शुरू किया गया जैसे इस प्रोजेक्ट के पुराने Python ड्राइवर उसे शुरू करते थे: `--bind`, `--metrics-bind`, `--target=` (forward-auth मोड में खाली, रिवर्स-प्रॉक्सी मोड में ऑरिजिन), `--difficulty 2`, `--slog-level ERROR`, `--cookie-secure=false`, और एक नीति जो हर `Mozilla` User-Agent को चैलेंज करती है, DNSBL और हनीपॉट अक्षम करके। `GOMAXPROCS` Sibuna के वर्करों की संख्या के बराबर है, और दोनों उत्पादों का हर थ्रेड उसी CPU समुच्चय पर पिन है। Hashcash कार्य बराबर रखा गया है: Sibuna के लिए 8 शून्य बिट, Anubis के लिए 2 शून्य हेक्साडेसिमल अंक, वही अपेक्षित 256 हैश। हार्नेस हर उत्पाद का सत्र वैसे ही प्राप्त करता है जैसे कोई ब्राउज़र करता। Anubis के लिए वह चैलेंज पेज लाता है, `<script id="anubis_challenge">` से चैलेंज पढ़ता है, वह नॉनस खोजता है जिसका `SHA-256(randomData || nonce)` `00` से शुरू होता है, और उसे पेज की सत्यापन कुकी के साथ `pass-challenge` को देता है। forward-auth मोड में `admitted` और `challenged` वर्कलोड Anubis के check endpoint (`/.within.website/x/cmd/anubis/api/check`) से पूछते हैं।

पूरे उत्पाद के वर्कलोड

चार वर्कलोड, लोड और आँकड़े पूरे-उत्पाद हार्नेस वाले ही हैं, एक बदलाव के साथ: ऑरिजिन `caddy respond` है, लोड CPU पर पिन किया गया, जैसे पुराने ड्राइवर उपयोग करते थे। हर रन से पहले हार्नेस तब तक प्रतीक्षा करता है जब तक `TIME_WAIT` में 3,000 से कम सॉकेट न रह जाएँ, क्योंकि Anubis के रिवर्स-प्रॉक्सी रन दसियों हज़ार सॉकेट पीछे छोड़ते हैं।

तालिका 18 Forward-auth मोड: पाँच-सेकंड के तीन `wrk` रन की माध्यिकाएँ (`benchmarks/results/comparison-latest.md`, 2026-10-16T00:55:19Z पर बनी)।

वर्कलोड	आँकड़ा	Sibuna Gate	Sibuna Shield	Anubis 1.27.0
admitted	req/s (स्टेटस)	210,682 (200)	208,635 (200)	36,509 (200)
	p99 us	305	308	7,477
	CPU us/req	12.09	13.21	106.30
	पीक RSS KiB	16,228	16,316	32,624
challenged	req/s (स्टेटस)	198,251 (401)	193,087 (401)	113,170 (401)
	p99 us	326	342	3,781
	CPU us/req	13.49	15.02	32.96
	पीक RSS KiB	16,240	16,328	32,696
allowed_static	req/s (स्टेटस)	212,569 (200)	212,444 (200)	89,904 (200)
	p99 us	304	304	4,547
	CPU us/req	11.73	11.95	41.63
	पीक RSS KiB	16,240	16,328	33,396
attack	req/s (स्टेटस)	212,086 (200)	211,347 (403)	36,853 (200)
	p99 us	305	306	7,283
	CPU us/req	12.07	12.56	104.88
	पीक RSS KiB	16,240	16,328	33,764

तालिका 19 रिवर्स-प्रॉक्सी मोड, `caddy respond` के पीछे: पाँच-सेकंड के तीन `wrk` रन की माध्यिकाएँ (`benchmarks/results/comparison-latest.md`, 2026-10-16T00:55:19Z पर बनी)।

वर्कलोड	आँकड़ा	Sibuna Gate	Sibuna Shield	Anubis 1.27.0
admitted	req/s (स्टेटस)	106,376 (200)	103,298 (200)	19,398 (200)
	p99 us	3,331	3,321	11,062
	CPU us/req	27.02	27.92	190.40
	पीक RSS KiB	16,528	16,444	39,980
challenged	req/s (स्टेटस)	198,402 (200)	197,327 (200)	28,842 (200)
	p99 us	325	328	9,546
	CPU us/req	13.05	13.78	137.73
	पीक RSS KiB	16,544	16,464	516,872
allowed_static	req/s (स्टेटस)	105,427 (200)	106,819 (200)	28,862 (200)

वर्कलोड	आँकड़ा	Sibuna Gate	Sibuna Shield	Anubis 1.27.0
	p99 us	3,320	3,316	30,124
	CPU us/req	26.24	26.52	135.41
	पीक RSS KiB	16,544	16,464	616,548
`attack`	req/s (स्टेटस)	105,616 (200)	210,483 (403)	23,399 (200)
	p99 us	3,327	306	14,124
	CPU us/req	26.98	12.28	169.04
	पीक RSS KiB	16,544	16,464	599,516

forward-auth मोड में Sibuna ने प्रति सेकंड 193,087 से 212,569 अनुरोध 304 से 342 us के p99 के साथ सर्व किए, और Anubis ने 36,509 (`admitted`) से 113,170 (`challenged`) 3,781 से 7,477 us के p99 के साथ। Sibuna के व्यस्त कोर चार में से 2.49 और 2.90 के बीच रहे, इसलिए सीमा लोड जनरेटर ने तय की; Anubis के 3.73 से 3.88 थे, इसलिए Anubis CPU से बँधा था। सबसे स्पष्ट आँकड़ा प्रति अनुरोध CPU समय है: एक स्वीकृत अनुरोध की लागत Sibuna Gate के लिए 12.09 us और Anubis के लिए 106.30 us थी। इस अंतर का एक संभावित भाग सत्र टोकन है। Anubis डिफ़ॉल्ट रूप से सत्रों पर Ed25519 से हस्ताक्षर करता है, और Sibuna एक कुंजी-युक्त BLAKE3 MAC जाँचता है (प्रिमिटिव तालिका में 155.8 ns); नीचे के प्रवेश मापन, जहाँ HS512 टोकन Anubis की सत्र जाँचों को आधे से अधिक बढ़ा देते हैं, इस व्याख्या का समर्थन करते हैं।

रिवर्स-प्रॉक्सी मोड में Sibuna की प्रॉक्सी की गई पंक्तियों ने 2.77 से 2.98 व्यस्त कोर पर प्रति सेकंड 103,298 से 106,819 अनुरोध सर्व किए। ये `product-latest.json` के स्टब ऑरिजिन वाले 138,965 से 147,303 से नीचे हैं, और डेमन CPU से बँधा नहीं है, इसलिए सीमा उसके बाहर है: ऑरिजिन, `caddy respond`, लोड CPU को `wrk` के साथ साझा करता है। इसकी टेल 2.1 से 2.4 ms के p90 में दिखती है। Anubis ने 3.68 से 3.97 व्यस्त कोर और प्रति अनुरोध 135 से 190 us CPU पर प्रति सेकंड 19,398 से 28,862 अनुरोध सर्व किए। उसकी resident मेमोरी `admitted` वर्कलोड में 39,980 KiB से बढ़कर `challenged` में 516,872 KiB और `allowed_static` में 616,548 KiB हो गई, और वर्कलोड के बाद उसने 589,276 KiB रखी, जबकि निष्क्रिय अवस्था में 21,552 KiB थी। बढ़ोतरी `challenged` में शुरू हुई, जहाँ हर अनुरोध को अपने id वाला एक नया चैलेंज मिलता है; यह प्रति जारी चैलेंज रखे गए किसी रिकॉर्ड का संकेत देता है, लेकिन हार्नेस ने कारण की पुष्टि नहीं की। Sibuna का पीक दोनों मोड की हर पंक्ति में 16,228 और 16,544 KiB के बीच रहा।

कई पंक्तियाँ भी एक जैसी स्थितियों की तुलना नहीं करतीं:

- forward-auth मोड में Anubis `/robots.txt` और `attack` पाथ का उत्तर स्वयं `200` से देता है, बिना ऑरिजिन के; Sibuna उनका उत्तर वैसे ही देता है जैसे उस मोड में किसी भी अनुरोध का।
- Anubis में अनुरोध निरीक्षण नहीं है, इसलिए वह `attack` वर्कलोड के SQL injection को प्रवेश देता है। Shield उसे अधिक कार्य के बाद `403` से अस्वीकार करता है; Gate उसे प्रवेश देता है, जैसे Anubis।
- उत्पाद सत्रों पर अलग-अलग तरह से हस्ताक्षर करते हैं (Anubis के लिए डिफ़ॉल्ट रूप से Ed25519, Sibuna के लिए कुंजी-युक्त BLAKE3 MAC), और वे अलग-अलग चैलेंज पेज रेंडर करते हैं।

प्रवेश ऑपरेशन

प्रवेश हार्नेस उन चार ऑपरेशनों का समय मापता है जिनसे प्रवेश बनता है, forward-auth मोड में, हर उत्पाद दो CPU पर (Sibuna के दो वर्कर, `GOMAXPROCS=2`)। लोड CPU पर दो क्लाइंट थ्रेड, हर एक एक keep-alive कनेक्शन के साथ, हर बैच को साझा करते हैं: 200 ऑपरेशन, या सत्यापन के लिए 64 प्रमाण, जो बैच से पहले तैयार किए जाते हैं। हम सात बैच की माध्यिका रिपोर्ट करते हैं।

- सत्र जाँच: वैध सत्र के साथ check endpoint (`200`)।
- अप्रमाणित जाँच: वही, बिना कुकी के (`401`)।
- चैलेंज बूटस्ट्रैप: चैलेंज शुरू करने के लिए ब्राउज़र जो लाता है। Sibuna के लिए यह दो अनुरोध हैं, इंटरस्टिशियल और चैलेंज रिकॉर्ड; Anubis के लिए एक, एम्बेडेड चैलेंज वाला पेज।
- प्रमाण सत्यापन: प्रति ऑपरेशन एक नया प्रमाण। Sibuna के प्रमाण वाले चैलेंज 25 ms के अंतर पर जारी होते हैं, उस प्रति सेकंड 50 से नीचे जिस पर उसकी अनुकूली कठिनाई बिट जोड़ती है, इसलिए बराबर रखा गया कार्य बना रहता है। Sibuna `--challenge-rate-limit 100000000` के साथ चलता है, इसलिए उसका लिमिटर जाँचा जाता है लेकिन कभी समाप्त नहीं होता।

Anubis दो बार चला, अपने डिफ़ॉल्ट Ed25519 टोकन के साथ और `--hs512-secret` के साथ।

तालिका 20 प्रति सेकंड प्रवेश ऑपरेशन, दो क्लाइंट, उत्पाद दो CPU पर (`benchmarks/results/admission-latest.md`, 2026-10-16T00:55:44Z पर बनी)

उत्पाद (टोकन)	सत्र जाँच	अप्रमाणित जाँच	चैलेंज बूटस्ट्रैप	प्रमाण सत्यापन	बाद में RSS, KiB
Sibuna (कुंजी-युक्त BLAKE3)	88,104	78,997	42,664	57,924	7,792
Anubis 1.27.0 (Ed25519)	10,951	28,619	7,893	12,668	33,400
Anubis 1.27.0 (HS512)	16,976	27,785	7,952	19,261	29,840

सात बैच की श्रेणियाँ फ़ाइल में हैं। Sibuna की श्रेणियाँ उसकी माध्यिकाओं के 11% के भीतर हैं; Anubis की अधिक चौड़ी हैं (Ed25519 टोकन के साथ उसकी अप्रमाणित जाँचें 25,394 से 32,741 तक फैली हैं), और 200 ऑपरेशन का एक बैच केवल 2 से 25 ms चलता है, इसलिए हम इन आँकड़ों को लगभग दसवें हिस्से की सटीकता तक पढ़ते हैं। धीमे क्लाइंट का प्रति ऑपरेशन समय round trip देता है: Sibuna सत्र जाँच के लिए 22.7 us, Ed25519 टोकन के साथ Anubis जाँच के लिए 182.5 us और HS512 के साथ 117.7 us।

ऊपर की किसी भी पंक्ति में Anubis ने Sibuna से अधिक अनुरोध सर्व नहीं किए, न कम CPU या मेमोरी उपयोग की। यह इस होस्ट पर इन वर्कलोड के बारे में एक कथन है, इससे अधिक कुछ नहीं।

> **Anubis की पंक्तियों को पढ़ना** Anubis एक सक्षम, व्यापक रूप से उपयोग होने वाला उत्पाद है, और ये पंक्तियाँ यह दावा नहीं हैं कि वह धीमा है। यह garbage-collected रनटाइम पर चलता है, डिफ़ॉल्ट रूप से सत्रों पर Ed25519 से हस्ताक्षर करता है ताकि सत्यापन के लिए किसी साझा सीक्रेट की ज़रूरत न हो, कई चैलेंज विधियाँ और एक समृद्ध नीति भाषा देता है, और चैलेंज की अवस्था एक स्टोर में रखता है; ये ऐसे डिज़ाइन चुनाव हैं जिनके लाभ यह हार्नेस नहीं मापता। पंक्तियाँ दिखाती हैं कि अक्टूबर 2026 में एक होस्ट पर एक लोड आकार के तहत हर उत्पाद को एक अनुरोध की कितनी लागत पड़ी। Sibuna नया है, और ये किसी स्थापित उत्पाद के साथ उसके पहले मापन हैं।

कंसोल का अतिरिक्त भार

SID 0007 कंसोल का पृथक्करण अनुबंध तय करता है। कंसोल सक्षम करने की लागत data-plane थ्रूपुट का अधिकतम 1% और p99 लेटेंसी का अधिकतम 10% हो सकती है, जिसे कॉन्फ़िगरेशन (कंपाइल से बाहर, कंपाइल में लेकिन अक्षम, निष्क्रिय, और आठ सक्रिय डैशबोर्ड) और वर्कलोड के एक मैट्रिक्स पर, बार-बार बारी-बारी से चले रन और रिपोर्ट की गई

अनिश्चितता के साथ मापा जाता है। अनिर्णायक या noisy रन अनुपालन स्थापित नहीं करते, और प्रिमिटिव आधार-रेखाएँ और smoke रन गेट का विकल्प नहीं हो सकते।

SID 0010 एक मापन दर्ज करता है: कंसोल कंपाइल होने पर, अक्षम, केवल स्टोरेज, या सक्षम और निष्क्रिय होने पर, 16 उत्पाद वर्कलोड उसके बिना बने डेमन के लगभग 1% के भीतर रहते हैं, जो noise के भीतर है। रिकॉर्ड इसके लिए कोई परिणाम फ़ाइल नहीं बताता। लोड में आठ सक्रिय डैशबोर्ड वाला कॉन्फ़िगरेशन मापा नहीं गया है, इसलिए SID 0007 का औपचारिक गेट खुला रहता है।

आठ-डैशबोर्ड रन कार्य की जाँच करता है, अतिरिक्त भार की नहीं। `sibuna-browser dashboards` ने एक SQLite नोड पर उसके कंसोल, चार वर्कर और एक ऐसी नीति के साथ रिलीज़ डेमन शुरू किया जो `/static/*`, `/about`, `/missing/*` और `/broken/*` की अनुमति देती है और बाकी सब को चैलेंज करती है; एक ऑरिजिन fixture `/missing/*` का उत्तर 404 से और `/broken/*` का 500 से देता है। एक जनरेटर ने 30 सेकंड तक प्रति सेकंड 200 अनुरोध भेजे, दस पाथ (एक SQL injection), तीन User-Agent और 500 फ़ॉरवर्ड किए गए पते बदलते हुए। हर डैशबोर्ड अपने सत्र के साथ अपने अलग Chromium में चला: डैशबोर्ड 1 से 3 एडमिनिस्ट्रेटर सत्र के रूप में, डैशबोर्ड 4 से 8 पहले वाले द्वारा जारी कोड से खुले wall display के रूप में, क्योंकि कंसोल प्रति पता और प्रति खाता प्रति मिनट पाँच साइन-इन की अनुमति देता है।

तालिका 21 एक नोड पर आठ Traffic डैशबोर्ड (`benchmarks/results/console-8-dashboards-2026-10-15.md`, 2026-10-15 को दर्ज, Chromium 155.0.8059.39)। परिणाम: PASS

माप	मान
Data-plane अनुरोध	30.0 s में 6,000 (प्रति सेकंड 200, 4 थ्रेड, 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)
शुरुआत में डेमन VmRSS	36,152 KiB
8 डैशबोर्ड खुले होने पर डेमन VmRSS	38,028 KiB
ट्रैफ़िक के बाद डेमन VmRSS	41,496 KiB (VmHWM 41,496 KiB)

ट्रैफ़िक के दौरान हर दो सेकंड पर और उसके 3 सेकंड बाद टाइलें पढ़ी गईं। हर डैशबोर्ड एक जैसे योग पर समाप्त हुआ (Requests 6,000, Admitted 2,400, Challenged 3,000, Policy denied 600, Origin 4xx 600, Origin 5xx 600), उसे 134 से 139 `stats` delta मिले, उसने अपना WebSocket खुला रखा, और कोई exception, कंसोल त्रुटि या Content-Security-Policy उल्लंघन रिपोर्ट नहीं किया। आठ ब्राउज़र subscriber कंसोल की 64 की सीमा से काफ़ी नीचे हैं। इस रन के डेमन में SQLite स्टोरेज और कंसोल है, उत्पाद हार्नेस के मेमोरी में चलने वाले डेमन के विपरीत, इसलिए इसकी resident मेमोरी उनसे तुलनीय नहीं है।

जो मापा नहीं गया

तालिका 22 यह भाग क्या नहीं मापता, और मापन की जगह क्या है

विषय	यहाँ क्यों नहीं मापा गया	इसकी जगह क्या है
SafeLine community edition 9.x	केवल Docker Compose स्टैक के रूप में आता है; होस्ट पर कोई Docker डेमन नहीं था, और डिटेक्टर इमेज ऐसा open-source बिल्ड नहीं है जिसे वहाँ कंपाइल किया जा सके	प्रकाशित, विक्रेता द्वारा रिपोर्ट किए गए तथ्य (सितंबर 2026 में जाँचे गए): पहचान दर 71.65% (balance) और 76.17% (strict), false positive 0.07% (balance) और 0.22% (strict); न्यूनतम होस्ट SSSE3 वाला Linux x86_64 या arm64, 1 CPU कोर, 1 GB RAM, 5 GB डिस्क, Docker 20.10.14+ और Compose 2.0+
Cloudflare WAF	एक होस्टेड सेवा; loopback मापन WAF को नहीं, नेटवर्क पाथ को मापेगा	प्रकाशित प्लान कोटा (सितंबर 2026 में जाँचे गए): कस्टम नियम Free 5, Pro 20, Business 100, Enterprise 1000; rate limiting नियम Free 1, Pro 2, Business 5, Enterprise 100
लोड में एक क्लस्टर	अभी कोई हार्नेस एक PostgreSQL डेटाबेस पर कई नोड नहीं चलाता	वे प्रोटोकॉल गुण जिन्हें SID 0011 निर्दिष्ट करता है (एक साझा सीड, जारीकर्ता से बँधे चैलेंज) और उसका तीन-नोड कार्यात्मक परीक्षण (मानदंड 2); कोई मापन नहीं
अनुरोध पाथ पर स्टोरेज	हर हार्नेस में दोनों डेमन मेमोरी में चलते हैं	SID 0011: एक स्टोरेज थ्रेड सारे SQL का स्वामी है, और अनुरोध पाथ कभी उसकी प्रतीक्षा नहीं करता
TLS, WAN, वास्तविक ब्राउज़र	हार्नेस loopback और HTTP का उपयोग करते हैं; सत्र हर स्थिति के लिए एक बार `sibuna-pow` से हल किए जाते हैं	कुछ नहीं; हर डिप्लॉयमेंट अपना पाथ स्वयं मापता है
ब्राउज़र सॉल्वर का हल-समय	हार्नेस नेटिव सॉल्वर से 8 बिट पर हल करता है	ऊपर दिया मॉड्यूल का आकार; भाग VII समझाता है कि नेटिव समय-माप ब्राउज़र के हल-समय का प्रतिनिधित्व क्यों नहीं करते
आवंटन की गिनती	हार्नेस एलोकेटर को instrument नहीं करता	भाग V के steady-state परीक्षण
लोड में आठ सक्रिय डैशबोर्ड के साथ कंसोल का पृथक्करण	SID 0007 का औपचारिक गेट नहीं चलाया गया है	SID 0010 का आँकड़ा (लगभग 1% के भीतर, कंसोल निष्क्रिय) और ऊपर का कार्यात्मक रन

इस तालिका की कोई पंक्ति उत्पाद के प्रदर्शन के बारे में दावा नहीं है। SafeLine की अपनी संख्याएँ पहचान की गुणवत्ता से संबंधित हैं, थ्रूपुट से नहीं, और Cloudflare प्रति-अनुरोध लागत के बजाय प्लान कोटा प्रकाशित करता है। Docker होस्ट वाला कोई भी व्यक्ति SafeLine को उन्हीं `wrk` वर्कलोड के पीछे चला सकता है।

अभ्यास 8.1. किसी दूसरी मशीन पर `cargo xtask bench primitives` चलाएँ और Hashcash सत्यापन तथा टोकन सत्यापन की पंक्तियों की तुलना संदर्भ होस्ट से करें। दोनों में से कौन हार्डवेयर हैश निर्देशों पर अधिक निर्भर है, और क्यों?

अभ्यास 8.2. forward-auth तालिका में किसी पंक्ति के व्यस्त कोर निकालें: उसके प्रति अनुरोध CPU माइक्रोसेकंड गुणा उसके प्रति सेकंड अनुरोध, दस लाख से भाग देकर। क्या डेमन CPU से बँधा था? फिर उसके प्रति अनुरोध CPU माइक्रोसेकंड की तुलना पहली तालिका की वर्गीकरण लागत से करें। प्रति अनुरोध CPU लागत इतनी अधिक क्यों है, और अंतर किन घटकों से आता है?
संकेत: `accept` और flush किए गए रिस्पॉन्स के बीच की वे सब चीज़ें सूचीबद्ध करें जिनका समय प्रिमिटिव सूट नहीं मापता।

अपरिवर्तनीय नियम समझाएँ. इस भाग का हर आँकड़ा दोहरावों पर माध्यिका है, उसके फैलाव के साथ। बैच की माध्यिका वाले आँकड़े और प्रति-अनुरोध p99.9 के बीच का अंतर समझाएँ, प्रिमिटिव तालिका पहला और लेटेंसी तालिका दूसरा क्यों रिपोर्ट करती है, और केवल दूसरा ही ऊपर वर्णित थ्रेड-turnover रुकावट को क्यों प्रकट कर सकता था।

भाग IX

संचालन और डिप्लॉयमेंट

हम दोनों सतहों, हर कमांड-लाइन फ़्लैग, *forward-auth* की विधियों, नीति फ़ाइलों, *SQLite* या *PostgreSQL* में स्थायी स्टोरेज, एक साझा डेटाबेस पर क्लस्टरिंग, और पैकेजिंग को समझाते हैं।

सतहें और कॉन्फ़िगरेशन

उद्देश्य इस अध्याय के अंत तक हमें यह कर पाना चाहिए: Sibuna को रिवर्स प्रॉक्सी या forward-auth सत्यापनकर्ता के रूप में चलाना, सतह चुनना, proof-of-work टियर और कठिनाई सेट करना, और मास्टर सीक्रेट का प्रबंधन करना।

सतह चुनना

- *Gate* (`--gate`, उपनाम `--no-waf`): proof-of-work प्रवेश, सत्र, घोषणात्मक नियम, प्रतिष्ठा, GCRA सीमाएँ, प्रतिबंध।
- *Shield* (डिफ़ॉल्ट, `--shield`, उपनाम `--waf`): Gate और साथ में सिमैंटिक फ़ायरवॉल।
- वितरित डिप्लॉयमेंट (दोनों में से किसी भी सतह के लिए एक विकल्प): `--data-dir` नीतियों, प्रतिष्ठा और फ़ॉरेंसिक जानकारी को एक स्थानीय SQLite फ़ाइल में रखता है; `--cluster-node` के साथ `--database-url` कई नोड को एक PostgreSQL डेटाबेस से जोड़ता है (SID 0011)।

कमांड-लाइन संदर्भ

तालिका `sibuna --help` के हर विकल्प की सूची देती है। सीमाएँ वही हैं जिन्हें पार्सर लागू करता है।

तालिका 23 डेमन के विकल्प

फ़्लैग	डिफ़ॉल्ट	अर्थ
`--port, -p`	`8080`	सुनने का पोर्ट, 1 से 65535
`--host, -h`	`0.0.0.0`	सुनने का पता, एक IP पते का लिटरल
`--upstream-host`	`127.0.0.1`	ऑरिजिन होस्ट (रिवर्स प्रॉक्सी मोड)
`--upstream-port, -u`	`3000`	ऑरिजिन पोर्ट
`--mode, -m`	`reverse_proxy`	`reverse_proxy` या `forward_auth` (`proxy` और `auth` भी), एक बार दिया गया
`--workers, -w`	प्रति CPU एक	listening सॉकेट साझा करने वाले accept थ्रेड, 0 (प्रति CPU एक) से 64; फिर हर स्वीकृत कनेक्शन को अपना थ्रेड मिलता है
`--max-connections`	`1024`	एक साथ सर्व किए जाने वाले कनेक्शन, 1 से 8192; इससे आगे वालों को `503` उत्तर मिलता है

फ़्लैग	डिफ़ॉल्ट	अर्थ
`--max-connections-per-ip`	`64`	एक सॉकेट पीयर पता कितने कनेक्शन रख सकता है, 0 (कोई सीमा नहीं) से 8192; IPv6 पीयर अपने `--ipv6-prefix` नेटवर्क के साथ गिना जाता है, और फ़ॉरवर्ड किए गए हेडर की कोई भूमिका नहीं होती; इससे आगे वालों को `503` उत्तर मिलता है
`--idle-timeout`	`15`	सेकंड में सबसे लंबी चुप्पी, 0 से 86400। अनुरोध हेड को इसके भीतर पहुँचना चाहिए। ऑरिजिन रिस्पॉन्स हर रीड पर इसे ताज़ा करता है, इसलिए धीमी स्ट्रीम तब तक चलती है जब तक बाइट बहते हैं, और चुप ऑरिजिन को दोनों सॉकेट पर काट दिया जाता है। प्रॉक्सी किए गए अपलोड को हर अवधि में 16 KiB पहुँचाने होंगे (slow-body हमलों के विरुद्ध न्यूनतम दर)। क्लाइंट सॉकेट पर इतनी ही लंबाई का write timeout होता है, इसलिए जो क्लाइंट इतनी देर तक एक भी बाइट स्वीकार नहीं करता, उसका कनेक्शन बंद कर दिया जाता है
`--websocket-idle-timeout`	`300`	अपग्रेड किए गए कनेक्शन की निष्क्रियता समय-सीमा सेकंड में, 0 से 86400
`--trust-forwarded`	बंद; forward-auth में चालू	पीयर से आए `X-Forwarded-For` / `X-Real-IP` को मानें
`--no-trust-forwarded`	बंद	इन्हें कभी न मानें, forward-auth मोड में भी नहीं
`--trusted-hops`	`1`	फ़ॉरवर्ड की गई पहचान पर भरोसा होने पर, क्लाइंट वह `X-Forwarded-For` प्रविष्टि है जो दाईं ओर से इतने स्थान पर है, 1 से 8
`--ja4-header`	बंद	वह अनुरोध हेडर जिसमें एक विश्वसनीय इनग्रेस क्लाइंट का JA4 TLS फ़िंगरप्रिंट भेजता है, जैसे `cf-ja4`; 1 से 64 token अक्षर, `X-Forwarded-*`, `X-Real-IP`, `X-Sibuna-*` या `Forwarded` नहीं। इसे केवल तभी पढ़ा जाता है जब फ़ॉरवर्ड की गई पहचान पर भरोसा हो (देखें "इनग्रेस से JA4 फ़िंगरप्रिंट")
`--ipv6-prefix`	`64`	IPv6 प्रीफ़िक्स की लंबाई, 32 से 128, जिस पर रेट लिमिट, प्रतिबंध और प्रति-क्लाइंट कनेक्शन सीमा लागू होती हैं
`--algorithm, -a`	`posw`	`posw` या `hashcash`
`--difficulty, -d`	`16`	वर्क बिट्स, 1 से 64: Hashcash के शून्य बिट, या PoSW गहराई जमा तीन
`--posw-challenges`	`16`	प्रति अनुक्रमिक-कार्य प्रमाण ओपनिंग, 1 से 32
`--token-scheme`	`mac`	`mac` (कुंजी-युक्त BLAKE3) या `ed25519`
`--token-ttl`	`86400`	सत्र की अवधि, सेकंड (1 से 31,536,000)

फ़्लैग	डिफ़ॉल्ट	अर्थ
`--challenge-ttl`	`300`	चैलेंज की अवधि, सेकंड (1 से 86,400)
`--secret-file, -s`	यादृच्छिक	64 हेक्स अक्षर या 32 कच्चे बाइट; `SIBUNA_SECRET` (64 हेक्स अक्षर) भी स्वीकार है
`--cookie-name`	`__sibuna_token`	सत्र कुकी का नाम, 1 से 64 token अक्षर
`--secure-cookie`	बंद	`Secure` एट्रिब्यूट जोड़ें
`--gate` / `--shield`	shield	सतह का चयन
`--rate-limit`	`100`	प्रति क्लाइंट प्रति विंडो अनुरोध (GCRA बर्स्ट)
`--rate-window`	`10`	सेकंड में विंडो, 1 से 86400
`--challenge-rate-limit`	`30`	प्रति क्लाइंट प्रति विंडो चैलेंज जारी करना और सत्यापन, अनुरोध बजट से अलग
`--ban-seconds`	`3600`	हनीपॉट प्रतिबंध की अवधि, 0 से 31,536,000
`--policy-file, -P`	कोई नहीं	घोषणात्मक JSON नीति
`--rate-table-cells`	`262144`	प्रति रेट लिमिटर टेबल सेल, हर एक 16 बाइट, 4,096 से 16,777,216, दो की अगली घात तक पूर्णांकित
`--ban-table-slots`	`65536`	बैन टेबल के स्लॉट, हर एक 24 बाइट, 1,024 से 16,777,216, दो की अगली घात तक पूर्णांकित
`--spent-tags`	`1048576`	रीप्ले के विरुद्ध याद रखे गए हल किए गए चैलेंज, हर एक 24 बाइट, 65,536 से 16,777,216
`--data-dir, -D`	कोई नहीं	एक नोड: SQLite फ़ाइल `<path>/sibuna.db` में सहेजें
`--database-url`	कोई नहीं	एक क्लस्टर: साझा PostgreSQL डेटाबेस (`postgres://` या `postgresql://`)
`--database-url-file`	कोई नहीं	डेटाबेस URL को केवल स्वामी द्वारा पठनीय फ़ाइल से पढ़ें
`--cluster-node`	कोई नहीं	इस नोड का id; चैलेंज कुंजियों और घटना पहचानकर्ताओं के ऊपरी बिट को बाँधता है; डेटाबेस URL के साथ आवश्यक
`--storage-poll-ms`	`500`	स्टोरेज tick का अंतराल, 1 से 60,000 ms
`--verbose, -v`	बंद	विस्तृत लॉगिंग
`--help` / `--version`		सहायता (कंसोल कंपाइल होने पर कंसोल विकल्प पहले) या संस्करण छापें, और बाहर निकलें

`--data-dir` और डेटाबेस URL एक-दूसरे को बाहर करते हैं, और `--database-url` तथा `--database-url-file` एक-दूसरे को बाहर करते हैं। `--cluster-node` के बिना डेटाबेस URL अस्वीकार किया जाता है। Zaxonlite क्लस्टर विकल्प `--cluster-listen`, `--cluster-peer`, `--cluster-secret-file` और `--cluster-tls-cert`, `--cluster-tls-key` तथा `--cluster-tls-ca` हटा दिए गए हैं; इनमें से कोई देने पर स्टार्टअप `RemovedOption` और SID 0011 का नाम देने वाले संकेत के साथ रुक जाता है। कंसोल के विकल्प अगले अध्याय में सूचीबद्ध हैं।

मास्टर सीक्रेट `--secret-file` या `SIBUNA_SECRET` के बिना डेमन एक यादृच्छिक सीड लेता है, और बैनर "random (set –secret-file)" छापता है। तब टोकन और चैलेंज प्रोसेस के साथ समाप्त हो जाते हैं, और जिस क्लस्टर के नोड अलग-अलग सीड रखते हैं वे एक-दूसरे की कुकी अस्वीकार करेंगे। `head -c 32 /dev/urandom | xxd -p -c 64 > /etc/sibuna/secret` से एक सीड बनाएँ और mode 0600 रखें।

कंसोल प्रीव्यू

कंसोल डिफ़ॉल्ट `console` feature द्वारा डेमन में कंपाइल होता है, लेकिन केवल `--console` के साथ शुरू होता है। इसे स्टोरेज चाहिए: `--data-dir` या `--database-url` के बिना `--console` को `CONSOLE001` (`StorageRequired`) के साथ अस्वीकार किया जाता है। इसे बाहर रखकर कंपाइल करने के लिए `cargo build --release -p sibuna --no-default-features --features sqlite` का उपयोग करें; केवल `--no-default-features` वाले बिल्ड में न स्टोरेज होता है न कंसोल। डेमन बंद रहते समय स्थानीय रूप से बूटस्ट्रैप करें:

```
sibuna init-admin admin --data-dir ./data
sibuna --data-dir ./data --console 127.0.0.1:19446
```

`sibuna console init-admin` वही कमांड है। `http://127.0.0.1:19446/console/` खोलें और बनाए गए अस्थायी पासवर्ड को बदलें, जिसे पहले साइन-इन पर बदलना अनिवार्य है और जो एक घंटे बाद समाप्त हो जाता है। साइन-इन के बाद का नेविगेशन घूमता हुआ देश-ग्लोब, अनुरोध और चैलेंज के आँकड़े, घटना जाँच, नीति और निरीक्षण संपादक, उपयोगकर्ता, सीमित दायरे वाले API टोकन, ऑडिट और सर्व करने वाले नोड के नियंत्रण देता है। प्रमाणीकरण वाला ढाँचा (shell) कोई ग्लोब ज्यामिति या टेलीमेट्री लोड नहीं करता। इंटरफ़ेस डेमन की बिल्ड स्क्रिप्ट द्वारा `cmd/console-ui` (क्रेट `sibuna-console-ui`) से बना एक WebAssembly मॉड्यूल है; यह और कमिट की गई stylesheet साधारण बिल्ड के साथ आते हैं।

तालिका 24 कंसोल के विकल्प

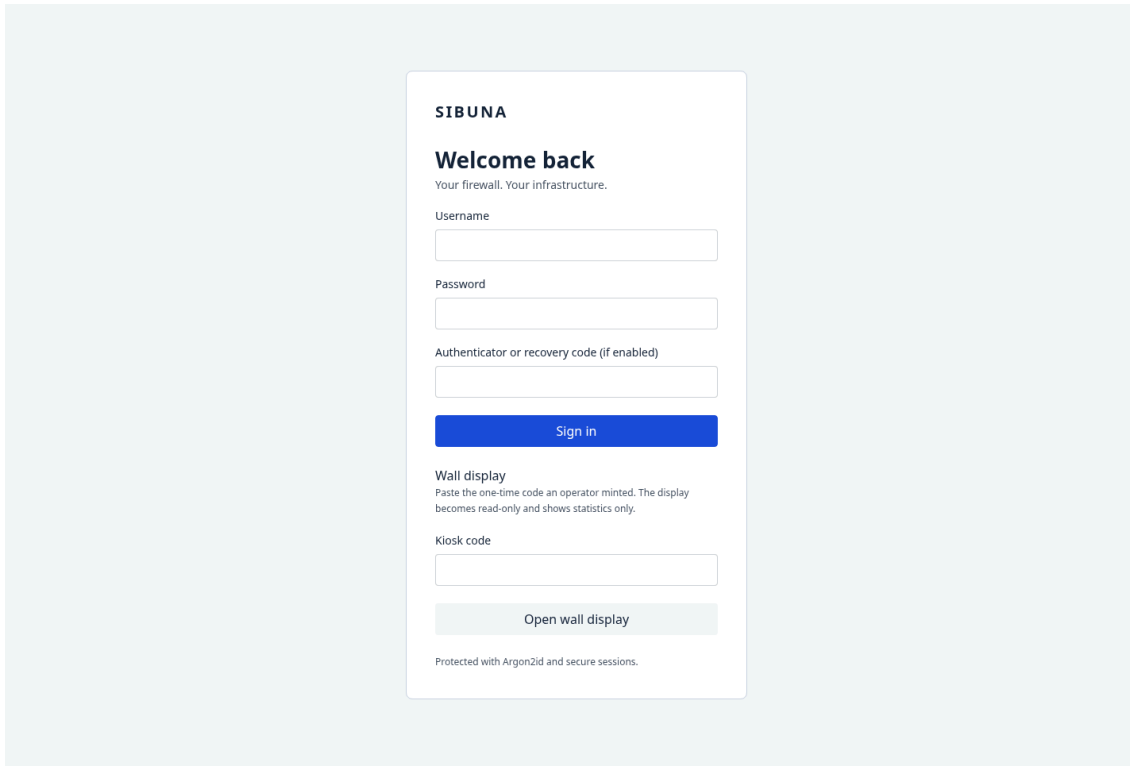
फ़्लैग	अर्थ
`--console <ip:port>`	इस listener पर कंसोल शुरू करें (IPv6 कोष्ठकों में, शून्य से भिन्न पोर्ट); `--data-dir` या `--database-url` आवश्यक
`--console-key-file <path>`	64 हेक्स अक्षर, केवल स्वामी की अनुमतियाँ; संग्रहीत second-factor और webhook सीक्रेट की रक्षा करता है
`--console-origin <https-origin>`	प्रॉक्सी के पीछे canonical HTTPS ऑरिजिन
`--console-behind-proxy`	TLS टर्मिनेट करने वाला एक प्रॉक्सी हर अनुरोध पहुँचाता है
`--console-trusted-proxy <cidr>`	ऐसा नेटवर्क जिसके कनेक्शन फ़ॉरवर्ड किए गए फ़ील्ड ले जा सकते हैं; दोहराया जा सकता है, अधिकतम 16
`--console-advertise <origin>`	वह लिंक जो दूसरे कंसोल इस कंसोल के लिए दिखाते हैं
`--console-location <latitude,longitude>`	ग्लोब पर इस नोड की घोषित स्थिति
`--console-probe <node-id>=<http://ip:port>`	health-check करने के लिए एक पीयर का data-plane listener; दोहराया जा सकता है, अधिकतम आठ

फ़्लैग	अर्थ
`--console-peer <node-id>=<https://origin>`	एक प्रबंधन पीयर; दोहराया जा सकता है, अधिकतम आठ
`--console-peer-key-file <path>`	पीयर कुंजी: केवल स्वामी, 64 हेक्स अक्षर, कंसोल कुंजी से स्वतंत्र
`--console-peer-ca-file <path>`	पीयर के लिए वैकल्पिक PEM trust anchor, जो सिस्टम roots की जगह लेते हैं
`--console-query-steps <n>`	प्रति aggregate रीड SQLite कदम, 100,000 से 50,000,000, डिफ़ॉल्ट 4,000,000
`--console-capture-heads`	प्रति घटना redact किए गए अनुरोध और ऑरिजिन रिस्पॉन्स हेड संग्रहीत करें; डिफ़ॉल्ट रूप से बंद
`--console-capture-header <name>`	संग्रहीत हेड में इस हेडर का मान रखें; दोहराया जा सकता है, अधिकतम 16; `--console-capture-heads` आवश्यक
`--console-cookie-secure`	प्रॉक्सी के बिना कंसोल कुकी को `Secure` एट्रिब्यूट दें (पार्सर स्वीकार करता है, `--help` में सूचीबद्ध नहीं)

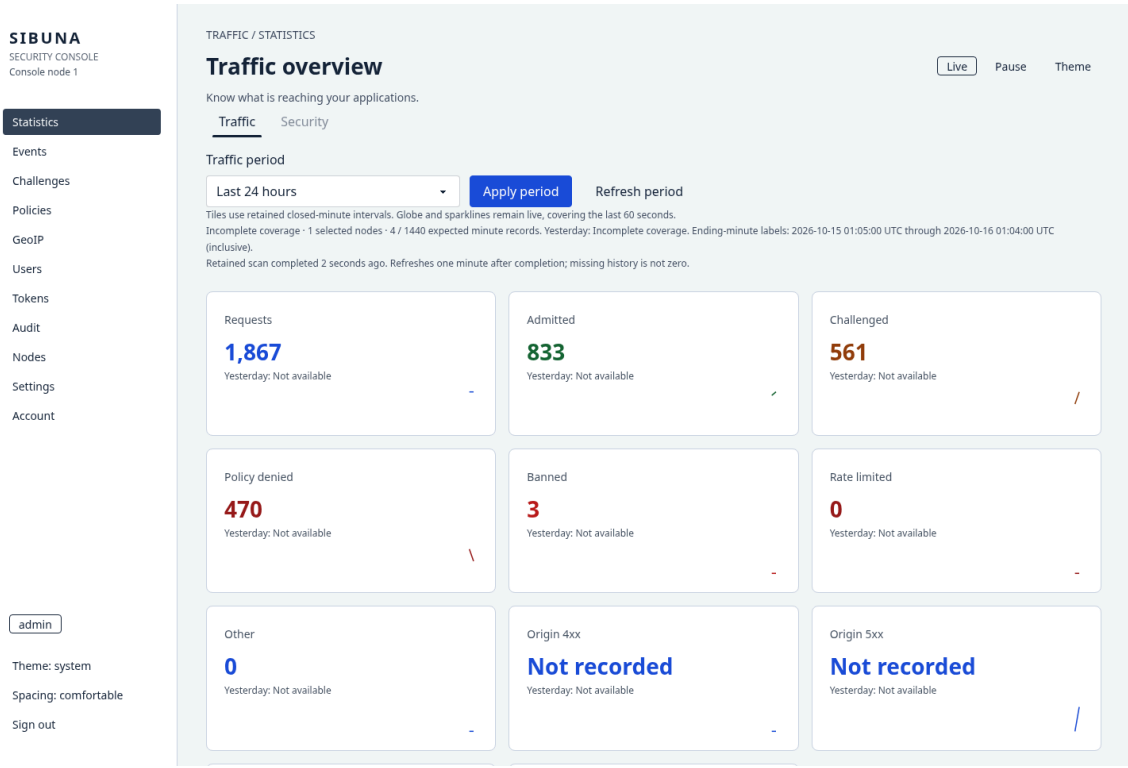
`--console` के बिना कंसोल विकल्प, दोहराया गया विकल्प या विकृत मान स्टार्टअप को `CONSOLE002` और विकल्प की त्रुटि के नाम के साथ रोक देता है। कंसोल के किसी नियम को तोड़ने वाला कॉन्फ़िगरेशन (उदाहरण के लिए विश्वसनीय प्रॉक्सी के बिना loopback से बाहर का listener) इसे `CONSOLE001` के साथ रोक देता है।

कंसोल की एक सैर

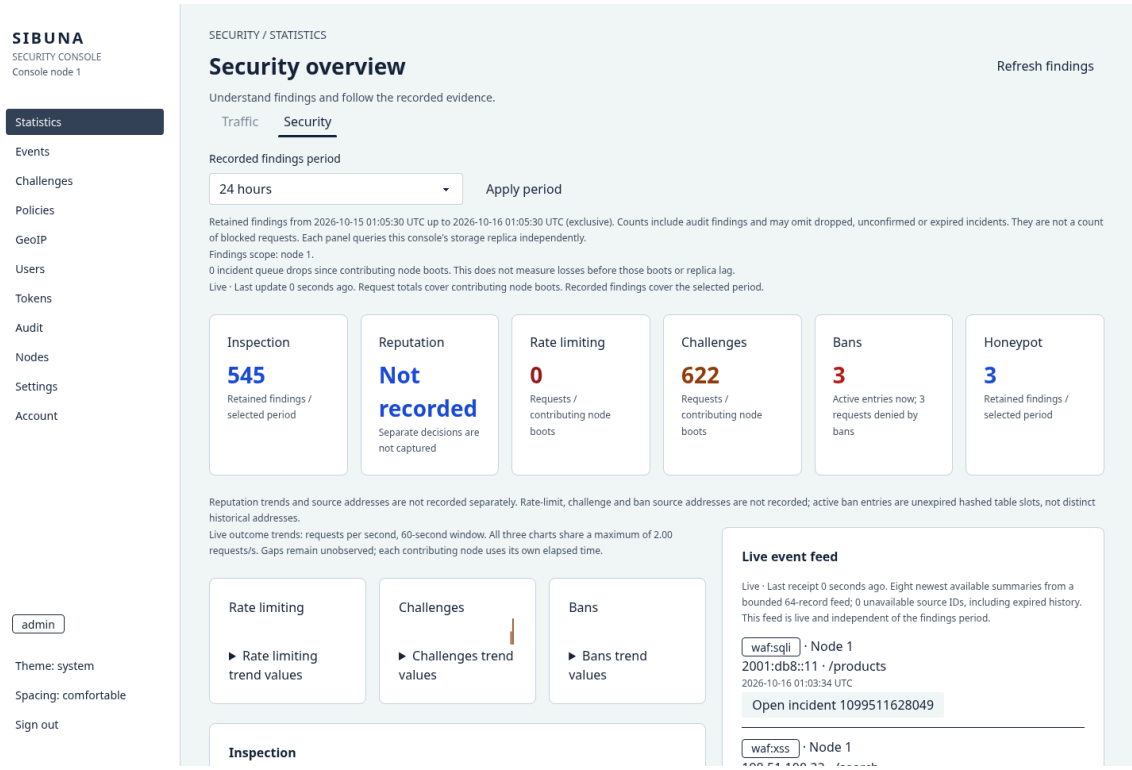
नीचे के स्क्रीनशॉट SQLite स्टोरेज वाले एक loopback नोड पर एक रिलीज़ बिल्ड से हैं, जिसकी नीति में डिफ़ॉल्ट चैलेंज के ऊपर चार `ALLOW` नियम हैं, लगभग चार मिनट के कृत्रिम ट्रैफ़िक के बाद: दस्तावेज़ीकरण श्रेणियों के पते, SQL injection और cross-site scripting वाले प्रोब, तीन हनीपॉट हिट और headless Chromium में हल किए गए सात चैलेंज। इसलिए हर संख्या केवल उदाहरण के लिए है। हमने इन्हें `cargo xtask browser console --viewport-only` से लिया, जो Chromium में 1440 x 960 पिक्सेल पर कंसोल में घूमता है और हर पेज को परखता है। हर पेज अपने शीर्षक में बताए गए एक प्रश्न का उत्तर देता है, और उत्पाद, सर्व करने वाला नोड, पेज और वापसी का रास्ता एक ही स्थान पर रखता है।



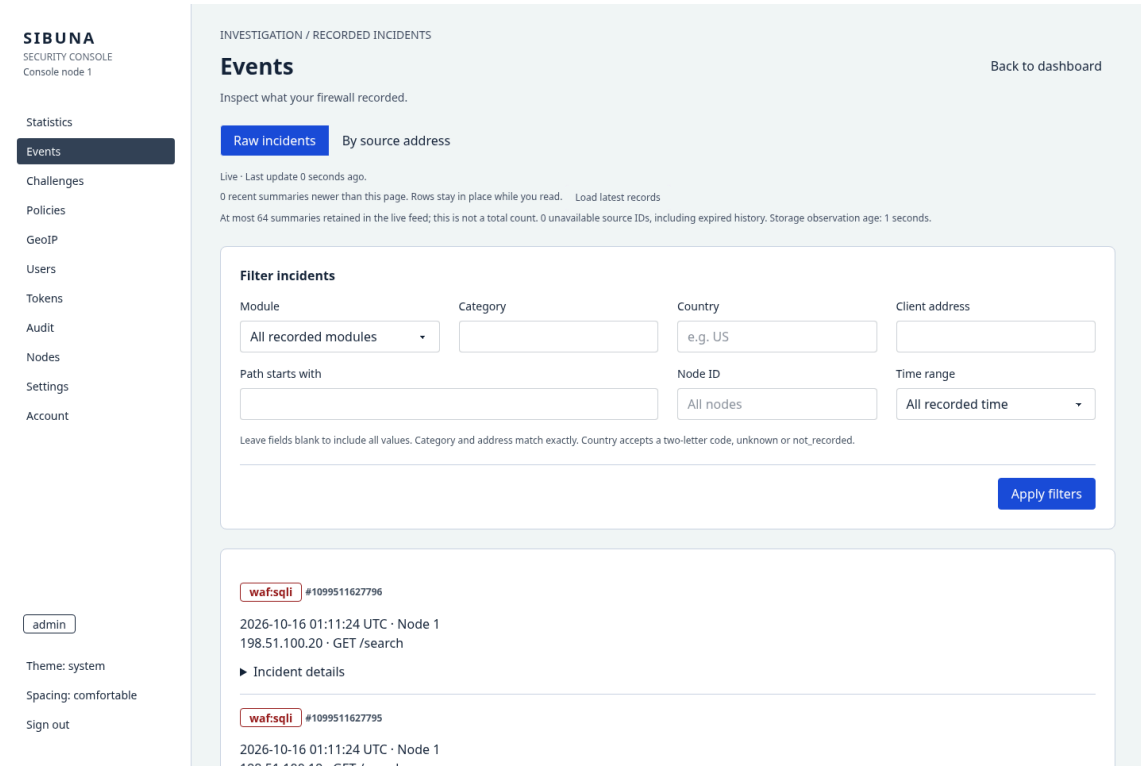
चित्र 13 साइन-इन। सार्वजनिक ढाँचा कोई ग्लोब ज्यामिति, GeoIP डेटा या टेलीमेट्री लोड नहीं करता; दूसरा फ़ॉर्म wall-display कोड के बदले केवल-पढ़ने वाला आँकड़ों का सत्र देता है।



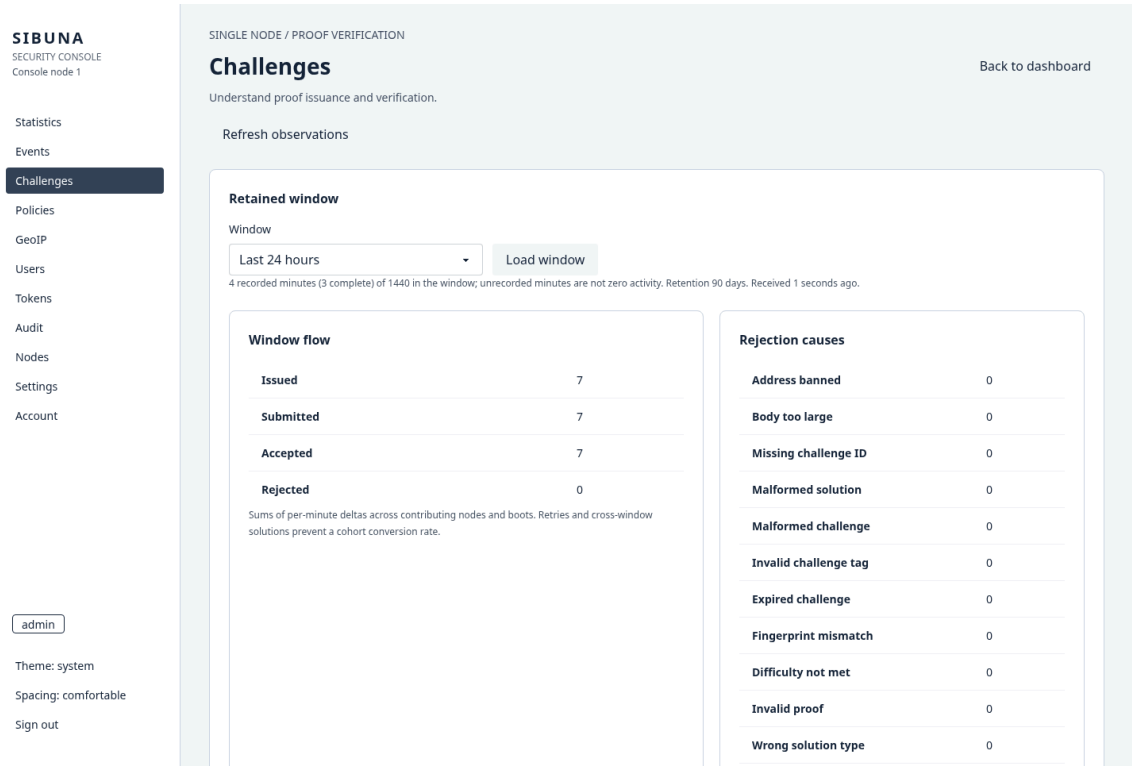
चित्र 14 आँकड़े, ट्रैफ़िक, पिछले 24 घंटों के। अवधि नियंत्रण के नीचे की टिप्पणी कवरेज बताती है: अपेक्षित 1,440 मिनट-रिकॉर्ड में से केवल चार मौजूद हैं, इसलिए हर "Yesterday" में "Not available" लिखा है। टाइलें रखे गए बंद मिनटों की गिनती एक लाइव 60-सेकंड sparkline के साथ दिखाती हैं; निर्णय के रंग (स्वीकृत हरा, चैलेंज किया गया एम्बर, अस्वीकृत लाल) हर पैनल में एक जैसे हैं। रखी गई अवधि के लिए ऑरिजिन 4xx और 5xx में "Not recorded" लिखा है, क्योंकि मिनट-रिकॉर्ड वह प्रॉक्सी मोड नहीं रखते जो ऑरिजिन गिनती को उसका अर्थ देता है; लाइव boot योग इन्हें दिखाते हैं। दिखने वाली टाइलों के नीचे दो स्तर-टाइलें हैं, सक्रिय प्रतिबंध प्रविष्टियाँ और स्वस्थ नोड, फिर समयरेखा, और सैंपल किए गए पैनल जो अनुरोध पाथ और referring होस्ट को क्रम देते हैं और क्लाइंट ऑपरेटिंग सिस्टम, ब्राउज़र और रिस्पॉन्स स्टेटस गिनते हैं।



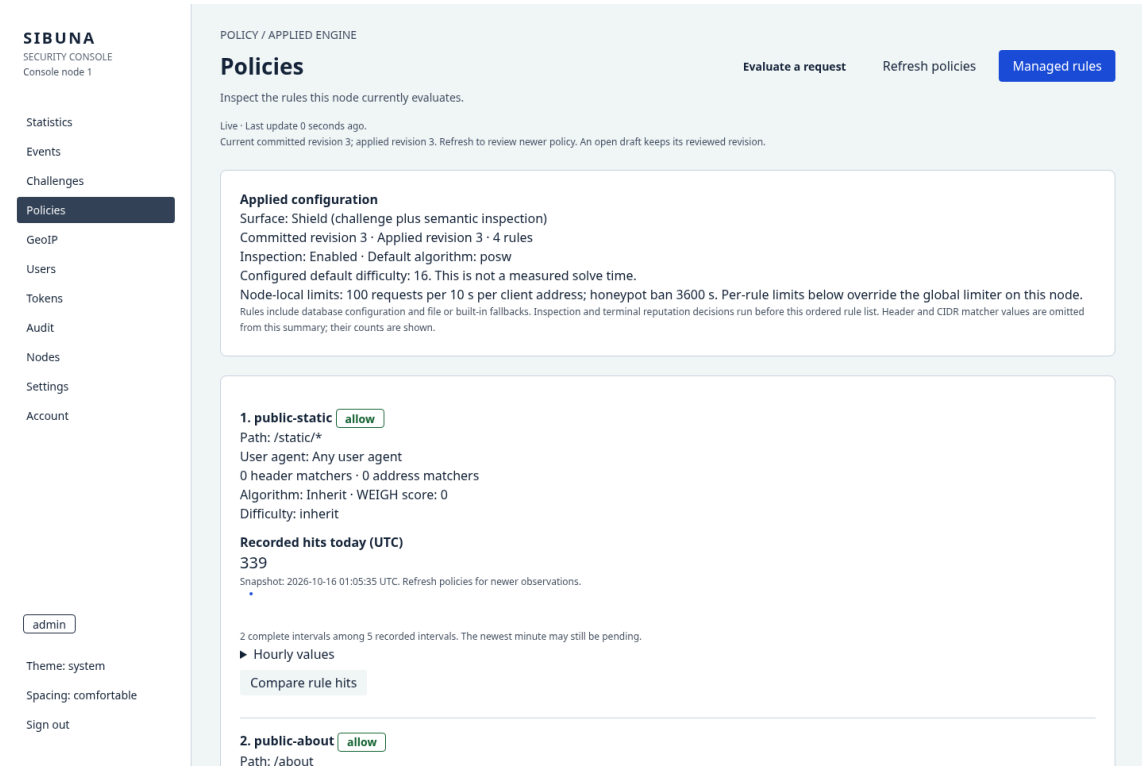
चित्र 15 आँकड़े, सुरक्षा। चुनी गई अवधि पर प्रति मॉड्यूल एक टाइल, लाइव परिणाम-रुझान, और सबसे नए निष्कर्षों के साथ लाइव इवेंट फ़ीड (यहाँ `waf:sqli` और `waf:xss`)। जो मान दर्ज नहीं होते, वे शून्य दिखाने के बजाय यही बताते हैं: Reputation टाइल में "Not recorded" लिखा है। पेज में और नीचे साठ बराबर बकेट पर निष्कर्ष, आक्रमण श्रेणियाँ और आक्रमण वाले पाथ हैं।



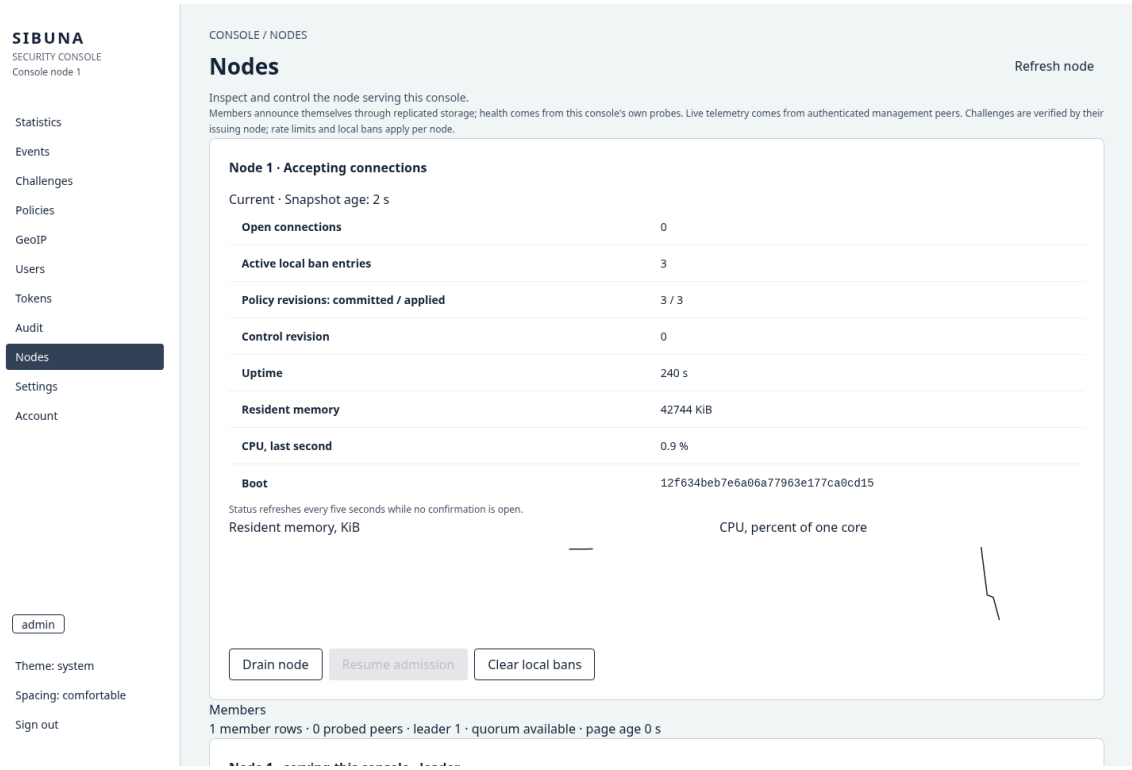
चित्र 16 इवेंट। दर्ज घटनाओं के ऊपर फ़िल्टर फ़ॉर्म, सबसे नई पहले। हर घटना अपने साक्ष्य के साथ वहीं खुलती है: चुना गया स्थानीय रिस्पॉन्स, बाइट की लंबाइयाँ, कैंपेन उम्मीदवार, और मिलान हुए नियम, स्कोर पदों और JA4 फ़िंगरप्रिंट के लिए स्पष्ट "not recorded" प्रविष्टियाँ। `--console-capture-heads` के साथ redact किया गया अनुरोध हेड (और रिवर्स प्रॉक्सी से गुज़रे ऑडिट किए गए प्रवेशों के लिए ऑरिजिन रिस्पॉन्स हेड) माँगने पर खुलता है, UTF-8 या Latin-1 के रूप में दिखाया जाता है, रिस्पॉन्स की स्थिति बताई जाती है (captured, local, forward-auth unobserved, origin unavailable), और एक copy-as-cURL कमांड होती है; केवल सूचीबद्ध हेडर मान रखे जाते हैं, और `--console-capture-header <name>` उस सूची में जोड़ता है। अन्यथा हेड "not recorded" के रूप में दिखते हैं। Deny और allow एक्शन IP समूह फ़ॉर्म को पते के साथ पहले से भरकर खोलते हैं।



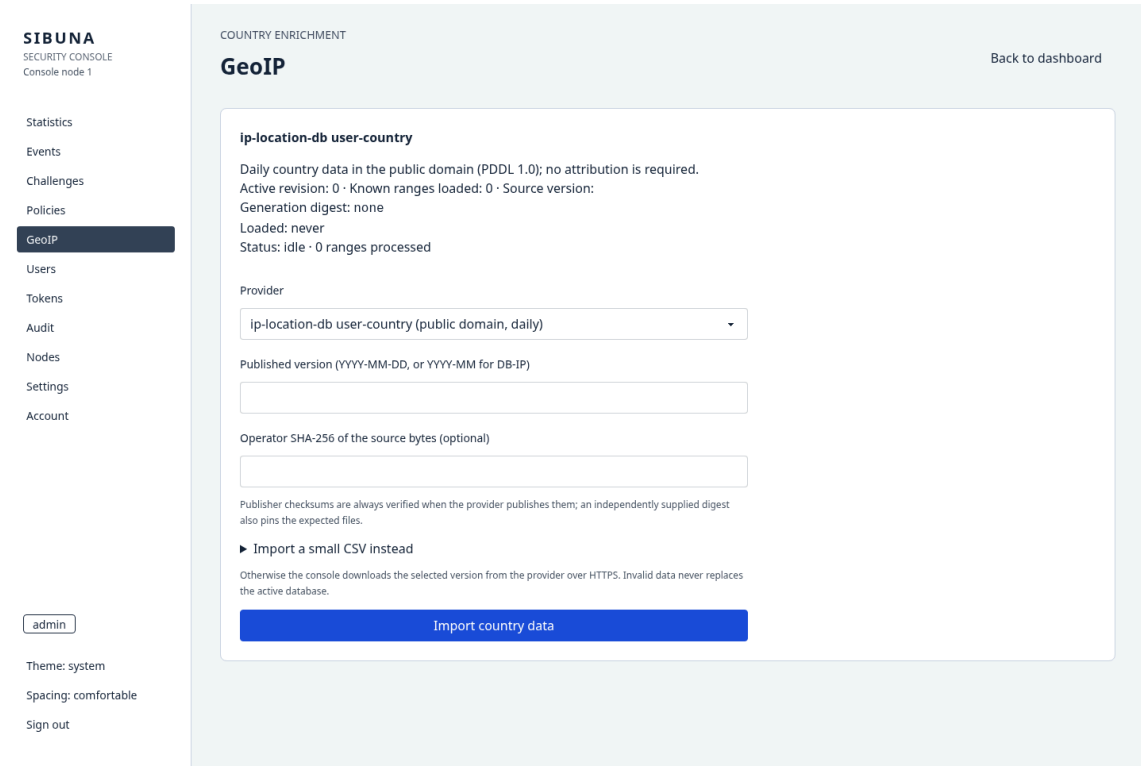
चित्र 17 चैलेंज। रखी गई विंडो (एक घंटा, एक दिन या सात दिन) स्थायी प्रति-मिनट रिकॉर्ड का योग करती है और बताती है कि कितने मिनट दर्ज और पूर्ण थे; यहाँ सात चैलेंज जारी, सबमिट और स्वीकार हुए, और अस्वीकृति के सभी कारण शून्य हैं। और नीचे इस boot के बाद के लाइव योग, कॉन्फ़िगर किए गए और सबसे हाल में जारी पैरामीटर, एल्गोरिदम और पैरामीटर-बिन के अनुसार बँटा स्वीकृत हल-समय का हिस्टोग्राम, अनुकूली कठिनाई के बदलाव, और deny तथा allow शॉर्टकट वाले प्रति-पता रिकॉर्ड हैं, जो सात दिन रखे जाते हैं।



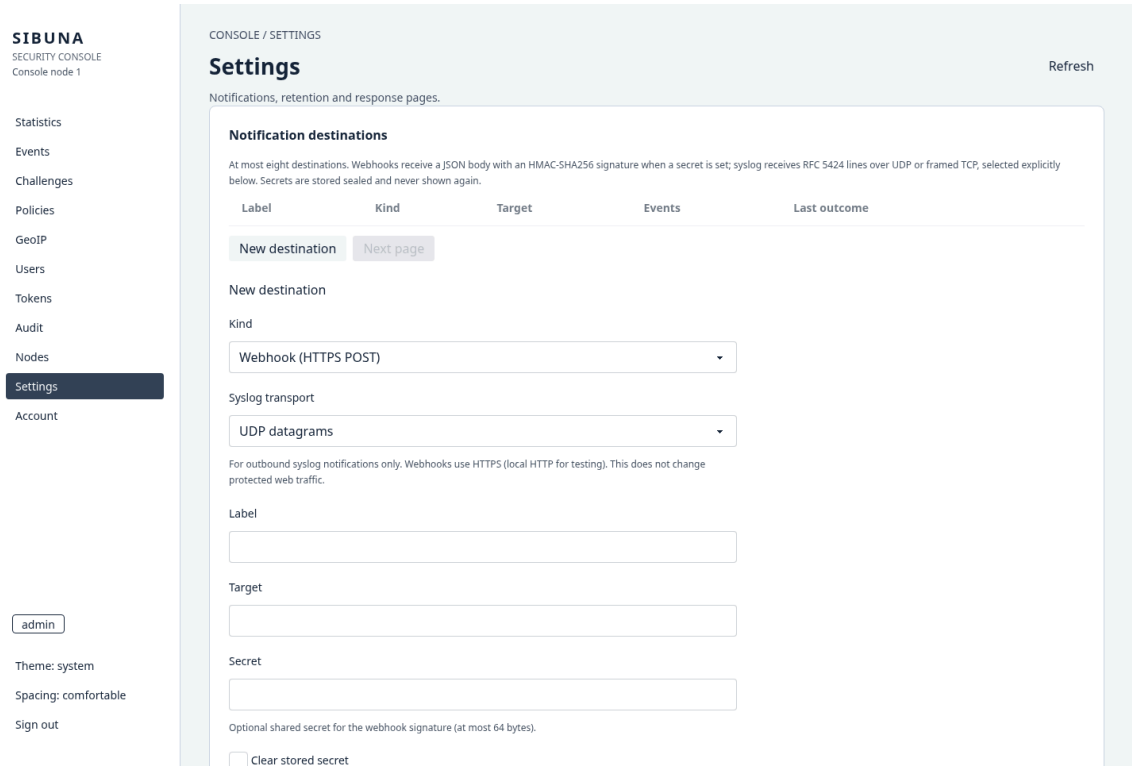
चित्र 18 नीतियाँ। लागू इंजन अपनी सतह, revision, डिफ़ॉल्ट एल्गोरिदम और कठिनाई और अपनी नोड-स्थानीय सीमाएँ बताता है, फिर मूल्यांकन के क्रम में नियमों को उनके आज के हिट और उन गिनतियों के कवरेज के साथ सूचीबद्ध करता है। और नीचे निरीक्षण मोड मैट्रिक्स, अनुरोध परीक्षक और IP समूह हैं। *Managed rules* संपादक खोलता है; किसी संपादन को सहेजने पर पहले बदले गए फ़ील्ड की समीक्षा दिखती है, और पुष्टि करने पर पूरे उम्मीदवार का सत्यापन होता है और एक नया revision बनता है।



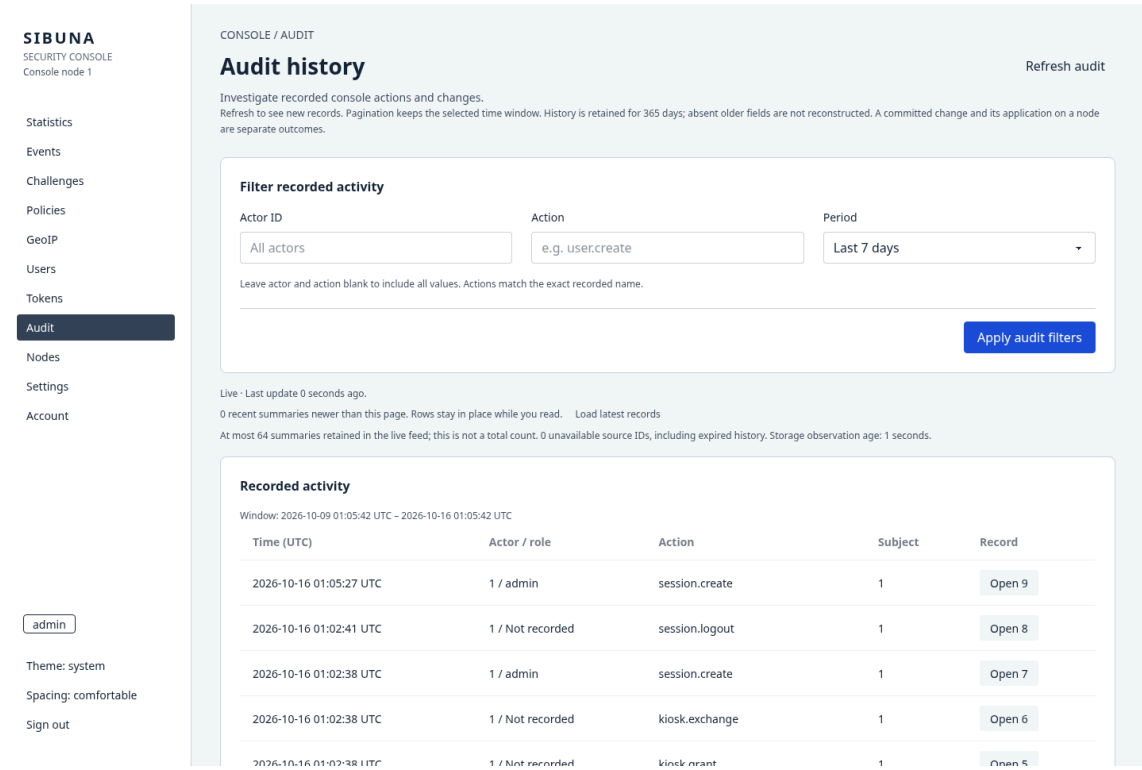
चित्र 19 नोड। सर्व करने वाले नोड की स्थिति और कमांड, पिछले सेकंड की resident मेमोरी और CPU तथा उनकी साठ-सेकंड sparkline के साथ; उनके नीचे हर घोषित सदस्य, कमिट किए गए revision के सामने उसका लागू नीति revision, उसका प्रबंधन पता और उसका probe परिणाम (SID 0007, SID 0013)।



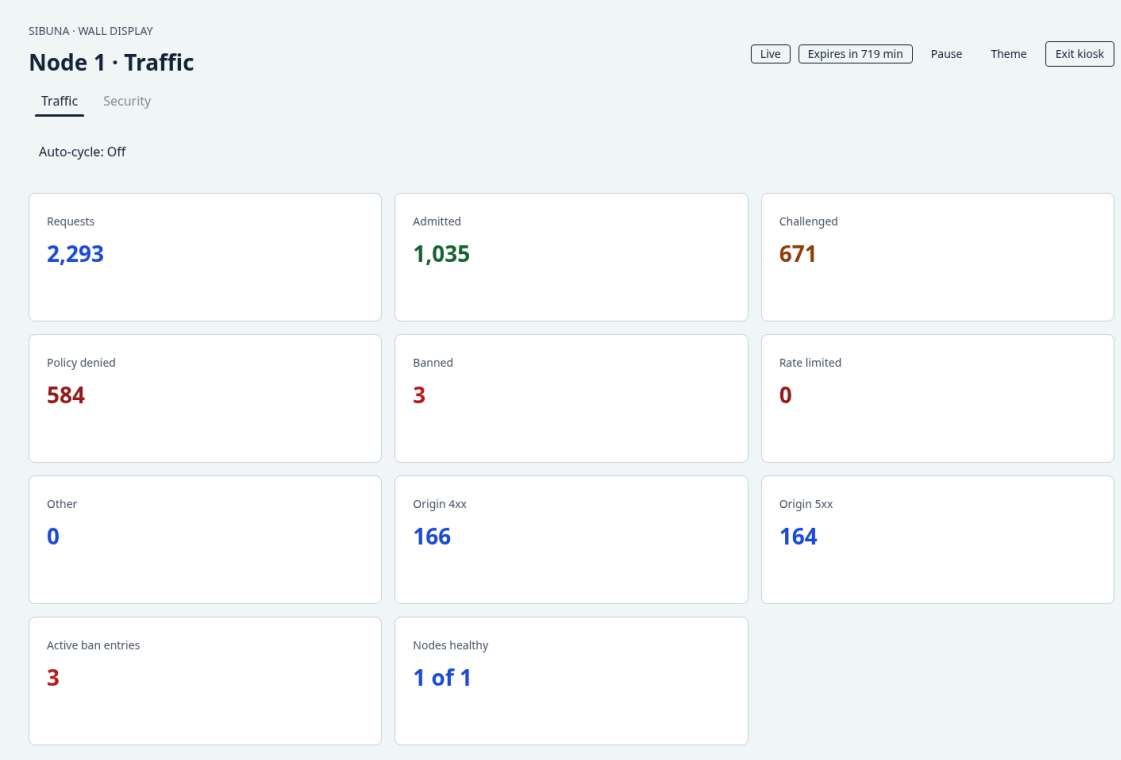
चित्र 20 GeoIP। प्रदाता, लाइसेंस, सक्रिय generation का digest, लोड का समय और आयात फ़ॉर्म; यहाँ अभी कोई generation लोड नहीं हुई है। विफल आयात कभी सक्रिय generation की जगह नहीं लेता।



चित्र 21 सेटिंग्स, जिनमें अभी कोई सूचना गंतव्य नहीं है और एक नए गंतव्य का फ़ॉर्म है। पेज में और नीचे अस्वीकृति-उछाल के थ्रेशोल्ड, प्रति विंडो स्पष्ट स्वीकृति के साथ प्रतिधारण (retention), रिस्पॉन्स-पेज टेम्पलेट और About हैं।



चित्र 22 ऑडिट। केवल जोड़े जा सकने वाला इतिहास, जिसमें कर्ता और भूमिका, एक्शन, विषय और redact किए गए पहले और बाद के सारांश वाला विवरण-दृश्य है; अस्वीकृत साइन-इन भी दर्ज होते हैं। हर प्रबंधन परिवर्तन और प्रमाणीकरण पंक्ति उस क्लाइंट पते को दर्ज करती है जिसने credential प्रस्तुत किया; सिस्टम द्वारा स्वयं लिखी गई पंक्तियाँ इसे "not recorded" दिखाती हैं। नीति रिकॉर्ड *Revert this change* देता है, जो पुष्टि के बाद उस revision से पहले दर्ज दस्तावेज़ को एक नए, ऑडिट किए गए revision के रूप में बहाल करता है, और यदि नियम समुच्चय आगे बढ़ चुका हो तो अस्वीकार हो जाता है।



चित्र 23 एक wall display। kiosk सत्र केवल Traffic और Security दृश्य दिखाता है, बिना नेविगेशन के, और बताता है कि वह कब समाप्त होगा।

वैकल्पिक `--console-location <latitude,longitude>` इस नोड की स्थिति घोषित करता है, उदाहरण के लिए Singapore में किसी डिप्लॉयमेंट के लिए `1.3521,103.8198`। ग्लोब शुरू में वहीं केंद्रित होता है; घुमाने के बाद Center Sibuna उसी बिंदु पर लौटाता है। देशों की गतिविधि नोड की ओर घूमते great-circle चापों के रूप में दिखती है, पिछले गोलार्ध और सपाट नक्शे की dateline पर कटाव के साथ। देशों की स्थितियाँ प्रतिनिधि केंद्रक (centroid) हैं, और तीर अलग-अलग कनेक्शन नहीं बल्कि देखी गई साठ-सेकंड की विंडो दर्शाते हैं। सेट न की गई सर्वर स्थिति स्पष्ट रूप से अज्ञात रहती है। कॉन्फ़िगर किया गया मार्कर टेलीमेट्री रुकने पर भी दिखता रहता है; पुराना ट्रैफ़िक एनिमेट नहीं होता।

नीति प्रीव्यू कॉन्फ़िगर किए गए फ़ाइल नियमों सहित एक निजी उम्मीदवार बनाते हैं। सहेजना अपेक्षित revision की तुलना करता है और फ़ील्ड तुलना की पुष्टि माँगता है। ऐतिहासिक revert वर्तमान नियम से तुलना करते हैं और एक नया revision बनाते हैं। कमिट किए गए और स्थानीय रूप से लागू revision अलग हैं। ऑडिट विवरण सीमित निर्णय-परिवर्तन और प्रभावी कार्यकारी भूमिका रखते हैं। matcher मान redact किए जाते हैं और अनुपस्थित संदर्भ वाले पुराने रिकॉर्ड स्पष्ट रूप से अनुपस्थित रहते हैं। Drain, resume और clear-local-bans के लिए कमांड प्रीव्यू आवश्यक है। जो कमांड तब आती है जब नोड पर कोई दूसरी कमांड चल रही हो, उसे कतार में लगाने के बजाय `503 CONSOLENODE` के साथ अस्वीकार किया जाता है। खोए हुए रिस्पॉन्स के बाद रीट्राई करने से पहले स्थायी रसीद (receipt) जाँचें; केवल कमिट हुआ इरादा यह स्थापित नहीं करता कि रनटाइम प्रभाव पूरा हो गया।

दूरस्थ संचालन के लिए एक HTTPS प्रॉक्सी का उपयोग करें और `--console-origin`, `--console-behind-proxy`, स्पष्ट `--console-trusted-proxy` CIDR, और `--console-key-file` कॉन्फ़िगर करें। कुंजी फ़ाइल में 64 हेक्साडेसिमल अक्षर होते हैं, उसकी अनुमतियाँ केवल स्वामी की होती हैं और वह संग्रहीत second-factor सीक्रेट की रक्षा करती है; इसे रीस्टार्ट के बाद भी, चैलेंज सीड से अलग, बनाए रखें। loopback से बाहर की पहुँच के लिए कॉन्फ़िगर किया गया HTTPS ऑरिजिन और प्रॉक्सी allowlist अनिवार्य हैं। Sibuna कंसोल listener पर भी कोई आने वाला TLS

टर्मिनेट नहीं करता; कंसोल TLS का उपयोग केवल अपने बाहर जाने वाले कनेक्शनों के लिए, rustls के ज़रिए, करता है (SID 0012)।
नेटिव CLI चल रहे कंसोल तक उन्हीं प्राधिकरण और स्टोरेज अनुबंधों से पहुँचता है। यह credential निजी फ़ाइलों से पढ़ता है, कमांड-लाइन मानों से नहीं:

```
sibuna console geoip update --version 2026-09-09 \
    --origin http://127.0.0.1:19446 --username admin \
    --password-file ./admin-password
```

केवल स्वामी द्वारा पठनीय पासवर्ड फ़ाइल का उपयोग करें, पहले पासवर्ड सेटअप पूरा करें, और आवश्यकता होने पर authenticator या recovery कोड के लिए `--factor-file` जोड़ें; खाते और GeoIP संचालनों के लिए `--token-file` उपयोगकर्ता नाम और पासवर्ड की जगह लेता है। क्लाइंट HTTP/1.1 बोलता है, लिटरल loopback पते को छोड़कर HTTPS माँगता है, कभी redirect का अनुसरण नहीं करता, और हर अनुरोध को 20 सेकंड तथा हर उत्तर को 16 KiB तक सीमित रखता है। `geoip status` सक्रिय generation पढ़ता है। अपडेट चुने गए प्रदाता की प्रकाशित फ़ाइलें HTTPS पर डाउनलोड करते हैं (डिफ़ॉल्ट रूप से public-domain `user-country` डेटासेट, या `--provider dbip --version YYYY-MM` के साथ DB-IP Lite), प्रकाशक के checksum (या `--checksum` से दिया गया checksum) सत्यापित करते हैं, `sibuna-geoip` के ज़रिए सीमित श्रेणियों का सत्यापन करते हैं, नई generation को सहेजते हैं और उसे स्थानीय रूप से सक्रिय करते हैं। विफल अपडेट पिछली generation को बनाए रखते हैं। CLI का टाइमआउट (`--timeout`, डिफ़ॉल्ट 1200 सेकंड) प्रतीक्षा रोकता है; वह सबमिट किए गए स्टोरेज कार्य को रद्द नहीं करता। DB-IP Lite को CC BY 4.0 attribution चाहिए, जो केवल तब दिखाया जाता है जब उसका डेटा सक्रिय हो। `SIBUNA_GEOIP_DATA=<snapshot>` वाला बिल्ड पहले स्थायी आयात तक एक सत्यापित स्नैपशॉट (`geoip-snapshot` टूल द्वारा लिखा गया) एम्बेड करता है; कंसोल के बिना बिल्ड इसे `GEOIP001` के साथ अस्वीकार करता है। अज्ञात पते, सैंपल की हानि और पुराना डेटा दिखते रहते हैं; देश डेटा आयात करने से न ट्रैफ़िक बनता है और न पहले ही समाप्त हो चुके सैंपल समृद्ध होते हैं।
क्लस्टर सदस्य साझा डेटाबेस की सदस्यता टेबल (`console_nodes`) के ज़रिए शुरू होते समय, हर मिनट, और लागू नीति या control revision बदलने के दो सेकंड के भीतर स्वयं की घोषणा करते हैं, और हर कंसोल के Nodes पेज पर दिखते हैं। कोई leader चुनाव नहीं है: SQLite नोड `single` भूमिका और स्वयं को leader बताता है, और PostgreSQL नोड `unknown` भूमिका और कोई leader नहीं बताता, क्योंकि PostgreSQL का failover Sibuna को दिखाई नहीं देता (SID 0007)। जिन पीयर के data-plane listener की health-check यह कंसोल करे उनके लिए `--console-probe <node-id>=<http://ip:port>` कॉन्फ़िगर करें, और दूसरे कंसोल जो लिंक दिखाएँ उसके लिए `--console-advertise <origin>`। सीधी लाइव टेलीमेट्री `--console-peer <node-id>=<https://origin>` (दोहराया जा सकता है, अधिकतम आठ) और एक समर्पित `--console-peer-key-file` का उपयोग करती है। पीयर के लिए `--console-behind-proxy` आवश्यक है। दोनों endpoint को एक-दूसरे को सूचीबद्ध करना होगा और विश्वसनीय HTTPS इनग्रेस का उपयोग करना होगा। पीयर कुंजी 64 हेक्स अक्षरों वाली केवल-स्वामी फ़ाइल है, जो कंसोल कुंजी और चैलेंज सीड से स्वतंत्र रूप से दी जाती है। पीयर rustls और `ring` provider के ज़रिए एक प्रमाणित TLS WebSocket पर बात करते हैं (SID 0012)। `--console-peer-ca-file` वैकल्पिक रूप से निजी प्रबंधन PKI के लिए PEM trust anchor देता है; अन्यथा क्लाइंट सिस्टम roots का उपयोग करता है। होस्ट-नाम सत्यापन हमेशा लागू होता है और बंद नहीं किया जा सकता: हर प्रबंधन ऑरिजिन का होस्ट, DNS नाम या IP पते का लिटरल, पीयर के प्रमाणपत्र में नामित होना चाहिए। अस्वीकृत प्रमाणपत्र पीयर को `rejected` चिह्नित करता है और एक `peer.rejected` ऑडिट पंक्ति लिखता है। कुंजी बदलने और CA फ़ाइल बदलने के लिए समन्वित रीस्टार्ट चाहिए; पीयर का प्रमाणपत्र इसके बिना नवीनीकृत किया जा सकता है। Nodes API और उसकी subscription रसीद की आयु, घड़ी का अंतर, boot बदलाव और सैंपलिंग की हानि रिपोर्ट करते हैं। अनुपस्थित अवलोकन अनुपलब्ध रहते हैं और कटे हुए मान पुराने रहते हैं; प्राप्त आँकड़े कभी किसी दूसरे नोड का अपना योगदान नहीं बनते। डैशबोर्ड का *Live traffic scope* एक नोड चुनता है या कॉन्फ़िगर किए गए नोड को जोड़ता है। *Node coverage and locations* अनुपस्थित, पुराने और घड़ी-अंतर वाले स्रोतों, अवलोकन के समय और

कॉन्फ़िगर किए गए गंतव्यों की पहचान करता है। देशों की रैंकिंग छोड़ी गई स्रोत पंक्तियों से आई अनिश्चितता दिखाती है। तीर अपना प्राप्त करने वाला नोड बनाए रखते हैं, और अनुपस्थित नोड स्थितियाँ गढ़ी नहीं जातीं। जब किसी योगदान देने वाले स्रोत के पास लगातार अंतराल न हो, तब संयुक्त दर अनदेखी रहती है। किसी एक नोड के रखे गए सेकंड और वर्तमान मिनट की पाथ रैंकिंग देखने के लिए उसे चुनें। दूरस्थ क्वेरी प्रमाणित पीयर कनेक्शन का उपयोग करती हैं; क्वेरी का उत्तर एक 4 KiB WebSocket संदेश में समाना चाहिए, इसलिए सर्व करने वाला कंसोल न समाने वाले पेज की पंक्तियाँ आधी कर देता है। अनुपस्थित पीयर अनुपलब्ध रहते हैं और इतिहास के कर्सर नोड रीस्टार्ट के पार नहीं जा सकते। मिनट का इतिहास अपना नोड चयन और प्रति-नोड पंक्तियाँ रखता है। नेविगेशन और मोबाइल हेडर चुने गए ट्रैफ़िक स्रोत से स्वतंत्र रूप से सर्व करने वाले कंसोल नोड की पहचान बताते हैं। स्थानीय कमांड प्रति नोड रहती हैं।

ट्रैफ़िक टाइलें चुने गए नोड के रखे गए बंद-मिनट रिकॉर्ड के पिछले 24 घंटों पर खुलती हैं। अवधि नियंत्रण एक घंटा, सात या नब्बे दिन, और लाइव boot योग भी देता है। Yesterday वाले विचलन के लिए हर चुने गए नोड से पूरा मेल खाता कवरेज चाहिए। ग्लोब और sparkline अब भी अपनी अलग लाइव 60-सेकंड विंडो का वर्णन करते हैं। रखा गया स्कैन पूरा होने के एक मिनट बाद ताज़ा होता है, और बदले जाने तक अपने पिछले मान और आयु दिखाता रहता है; स्रोत या अवधि बदलने पर पुराना दायरा हटा दिया जाता है। अनुपस्थित इतिहास कभी शून्य नहीं बनता।

Compare retained traffic दो बंद मिनट-विंडो को अगल-बगल दिखाता है: कल की वही विंडो, पिछली अवधि, या कोई दूसरा नोड। अवधि और end-offset नियंत्रण दोनों UTC श्रेणियों को स्थिर कर देते हैं। हर एक्शन हर ओर अधिकतम सोलह सघन पेज पढ़ता है, प्रति पेज 96 संग्रहीत रिकॉर्ड के साथ। इससे बीच में रीस्टार्ट न होने पर 24 घंटे की विंडो एक बैच में आ जाती है; लंबे स्कैन के लिए Continue वही सीमाएँ बनाए रखता है। पूरा कवरेज, अधूरा कवरेज और अनुपस्थित इतिहास स्पष्ट रहते हैं। दर के प्रतिशत के लिए बराबर UTC विंडो में पूरे, न-अतिव्यापी अंतराल चाहिए, जो देखे गए मिलीसेकंड से सामान्यीकृत होते हैं; नए ट्रैफ़िक के साथ शून्य संदर्भ "New" दिखाता है। मुख्य Traffic अवधि स्वतंत्र रहती है। मिनट-प्रारूप ऐतिहासिक प्रॉक्सी मोड दर्ज नहीं करता, इसलिए forward-auth के शून्य को देखे गए रिस्पॉन्स मानने के बजाय ऑरिजिन-रिस्पॉन्स की तुलनाएँ अनुपलब्ध रहती हैं।

Compare retained path rankings दो बंद विंडो या नोड चुनता है। Node 0 में सभी रखे गए नोड शामिल हैं, सेवानिवृत्त सदस्यों सहित। एक एक्शन अधिकतम सोलह अपरिवर्तनीय archive पढ़ता है, उनकी पहचान और checksum जाँचता है, और प्रदर्शन की पंक्तियाँ चुनने से पहले हर काउंटर को मिलाता है। archive 60-सेकंड की late-sample विंडो के बाद सील होते हैं, इसलिए सबसे नया बंद मिनट अभी लंबित हो सकता है। Continue मूल विंडो और कर्सर बनाए रखता है। सीमाएँ सैंपल किए गए पाथ प्रीफ़िक्स का वर्णन करती हैं, अनुरोधों के सटीक योग का नहीं। पेज आंशिक स्कैन, truncation, प्रतिधारण की सीमाएँ और रिपोर्ट की गई कतार-हानि दिखाता है; अनुपस्थित archive न शून्य ट्रैफ़िक स्थापित कर सकते हैं न पूरा कवरेज।

Security पिछले 24 घंटों पर खुलता है और चुनी गई अवधि पर समय के साठ बराबर बकेट उपयोग करता है। इसके aggregate स्टोरेज थ्रेड पर एक कार्य-सीमा के अधीन चलते हैं। SQLite पर सीमा `--console-query-steps` virtual-machine कदम है (100,000 से 50,000,000, डिफ़ॉल्ट चालीस लाख)। इसे `--console` के साथ सेट करें; अमान्य या दोहराए गए मान अस्वीकार किए जाते हैं। PostgreSQL पर सीमा रीड के ट्रांज़ैक्शन में सेट 2,000 ms का statement timeout है (SID 0013); `--console-query-steps` बदलने से यह नहीं बदलता। सीमा समाप्त कर चुकी क्वेरी एक छोटी अवधि माँगती हैं। कदमों की गिनती क्वेरी के कार्य को सीमित करती है, बीते समय को नहीं। चार्ट एक पैमाना साझा करते हैं; UTC अंतराल की शुरुआत और समूहित सटीक गिनती के लिए Trend values खोलें। ये रखे गए निष्कर्ष हैं, इसलिए घटना कवरेज के अभाव को आक्रमणों का अभाव नहीं समझा जा सकता।

Wall display kiosk सत्र का उपयोग करते हैं। Account के अंतर्गत ऑपरेटर डिस्प्ले को नाम देता है और Create display code चुनता है। डिस्प्ले दस मिनट के भीतर एक-बार वाला कोड साइन-इन फ़ॉर्म में चिपकाता है; उसका सत्र केवल-पढ़ने वाला होता है, आँकड़ों तक सीमित होता है और बारह घंटे के भीतर समाप्त हो जाता है। खाता पेज नेविगेशन पर या छिपाए जाने पर दिखाया गया कोड मिटा देता है। छिपाने से बिना उपयोग हुई अनुमति (grant) रद्द नहीं होती; कोड बदलने (exchange) की समय-सीमा फिर भी लागू रहती है। कोड कभी URL में नहीं होने चाहिए। Traffic और Security डिस्प्ले साझा करते हैं: Security घटना के पते या पेलोड साक्ष्य के बिना मॉड्यूल के समग्र रुझान और अनुरोध परिणाम

दिखाता है। स्वचालित चक्र वैकल्पिक है, शुरू में बंद रहता है, और reduced motion, पुराने डेटा या Pause पर रुक जाता है।

सूचना गंतव्य (अस्वीकृति-उछाल, प्रतिबंध, पहुँच से बाहर सदस्य और leader बदलाव के लिए हस्ताक्षरित webhook और syslog) Settings के अंतर्गत हैं; webhook सीक्रेट के लिए `--console-key-file` चाहिए। leader-change प्रकार परिभाषित रहता है ताकि संग्रहीत गंतव्य अपना अर्थ बनाए रखें, लेकिन कोई नोड इसे नहीं उठाता, क्योंकि कोई leader चुनाव नहीं है। हर गंतव्य का अपना cooldown और तीन प्रयासों का रीट्राई बजट है। डिलीवरी ऑडिट रिकॉर्ड परिणाम दिखाते हैं; पूर्ण कतार का इतिहास सात दिन और 4,096 इवेंट तक सीमित है। webhook में एक स्थिर `Idempotency-Key` होता है ताकि प्राप्तकर्ता अनिश्चित नेटवर्क पूर्णता के बाद दोहराए गए प्रभावों को दबा सकें। webhook HTTPS पर भेजे जाते हैं, या सादा HTTP केवल लिटरल loopback पते पर, और उत्तर की बॉडी 64 KiB तक पढ़ी जाती है। syslog का UDP/TCP चयन केवल बाहर जाने वाली सूचनाओं पर लागू होता है, संरक्षित वेब ट्रैफ़िक से स्वतंत्र। मैनुअल परीक्षण इरादे और पूर्णता को Audit में दर्ज करते हैं और गंतव्य का परिणाम ताज़ा करते हैं; ऑपरेटर के रीट्राई करने से पहले अपुष्ट ऑडिट पूर्णता स्पष्ट रूप से दिखाई जाती है।

Settings प्रतिधारण को भी नियंत्रित करता है: मिनट-इतिहास के लिए एक से 90 दिन, रैंकिंग के लिए एक से सात, घटनाओं के लिए एक से 30 और ऑडिट के लिए एक से 365। ऊपरी मान डिफ़ॉल्ट हैं; रैंकिंग अपना 512 MiB कोटा रखती हैं। प्रतिधारण मान सहेजने के लिए पुष्टि चाहिए, क्योंकि सफ़ाई पुराने रिकॉर्ड को सीमित बैच में स्थायी रूप से हटा देती है। बाद में मान बढ़ाने से हटाया गया इतिहास वापस नहीं आता। पुराने संपादन अस्वीकार होते हैं और सफल बदलाव Audit में दिखते हैं।

रिस्पॉन्स पेज (challenge, denied, rate limited, banned, overloaded) Settings के अंतर्गत निश्चित placeholder वाले सीमित HTML के रूप में संपादित किए जा सकते हैं; ड्राफ़्ट एक sandboxed टैब में प्रीव्यू होते हैं और सहेजे गए पेज अगले नीति स्नैपशॉट से दिए जाते हैं।

Policies पेज पर नीति वर्कफ़्लो: managed नियमों का क्रम बदलें, किसी ड्राफ़्ट को रखे गए निरीक्षण निष्कर्षों के विरुद्ध दोबारा चलाएँ, IP समूहों और सक्रिय GeoIP generation से जुड़े देश-ब्लॉक का प्रबंधन करें, और managed समुच्चय को निर्यात या परमाणु रूप से आयात करें (`sibuna console policies export` और `sibuna console policies import --file <set.json>` भी)। एक आयात एक अनुरोध है जिसमें 1 से 128 दस्तावेज़ होते हैं, हर एक अधिकतम 4 KiB और कुल मिलाकर अधिकतम 256 KiB; लगभग 96 KiB से बड़े समुच्चय के लिए पेज की टिप्पणी कमांड-लाइन आयात का नाम देती है। लागू नियम आज के दर्ज हिट, प्रति घंटा sparkline और एक सुलभ तालिका दिखाते हैं। हिट एक सफल घोषणात्मक matcher मूल्यांकन है: मेल खाते WEIGH नियम और पहला अंतिम मैच गिने जाते हैं; पहले का कोई निरीक्षण या प्रतिष्ठा निर्णय मूल्यांकन को रोक सकता है। निजी परीक्षण इन काउंटरों को नहीं बढ़ाते। *Compare rule hits* एक दर्ज नोड के लिए दो बंद अवधियों को स्थिर करता है। revision इतिहास *Compare hits around this edit* देता है, जो संपादन वाले मिनट को छोड़ देता है और चुनी गई अवधि तक बराबर उपलब्ध अवधियों का उपयोग करता है। वैकल्पिक applied-revision फ़िल्टर असंबंधित generations को तुलना से बाहर रखते हैं। स्टार्टअप, cutover, छूटे हुए राइट और प्रतिधारण कवरेज में दिखने वाले अंतराल छोड़ते हैं; प्रतिशत के लिए दोनों ओर पूरा कवरेज चाहिए। नियम अवलोकन और प्रति घंटा/दैनिक सारांश मिनट-प्रतिधारण सेटिंग का अनुसरण करते हैं। किसी संपादन के आसपास देखा गया बदलाव एक तुलना है, इस बात का साक्ष्य नहीं कि संपादन ने ट्रैफ़िक बदला।

बाद की GeoIP generation आयात करने से देश से निकली मौजूदा प्रतिष्ठा पंक्तियाँ अपने आप ताज़ा नहीं होतीं। जोड़े गए, रखे गए और हटाए गए प्रीफ़िक्स की तुलना के लिए देश एक्शन का प्रीव्यू देखें; लागू करने से पहले समीक्षित diff के सभी पेज देखें। प्रतिस्थापन उस देश के स्वामित्व वाली अप्रचलित पंक्तियाँ हटाता है और स्वतंत्र रूप से प्रबंधित प्रीफ़िक्स को बनाए रखता है। बदली हुई generation या नीति revision के लिए नया प्रीव्यू चाहिए, और अतिव्यापी स्वतंत्र संपादन अस्वीकार किए जाते हैं। Events पेज में रखे गए WAF निष्कर्ष और हनीपॉट घटनाएँ हैं; यह पूरा access log नहीं है। GeoIP लोड होने पर स्टोरेज थ्रेड हर घटना का देश और generation घटना बैच के ट्रांज़ैक्शन में दर्ज करता है। यह सहेजते समय किया गया attribution है, अनुरोध के समय का पुनर्निर्मित स्थान नहीं। बाद के आयात दर्ज मैपिंग को नहीं बदलते। देश फ़िल्टर बड़े अक्षरों वाला दो-अक्षरी कोड, लोड की गई generation में अनुपस्थित पते के लिए `unknown`, या मैपिंग डेटा के बिना घटना के लिए `not_recorded` स्वीकार करता है। स्रोत समूह मिश्रित देश या कवरेज को स्पष्ट रूप से बताते

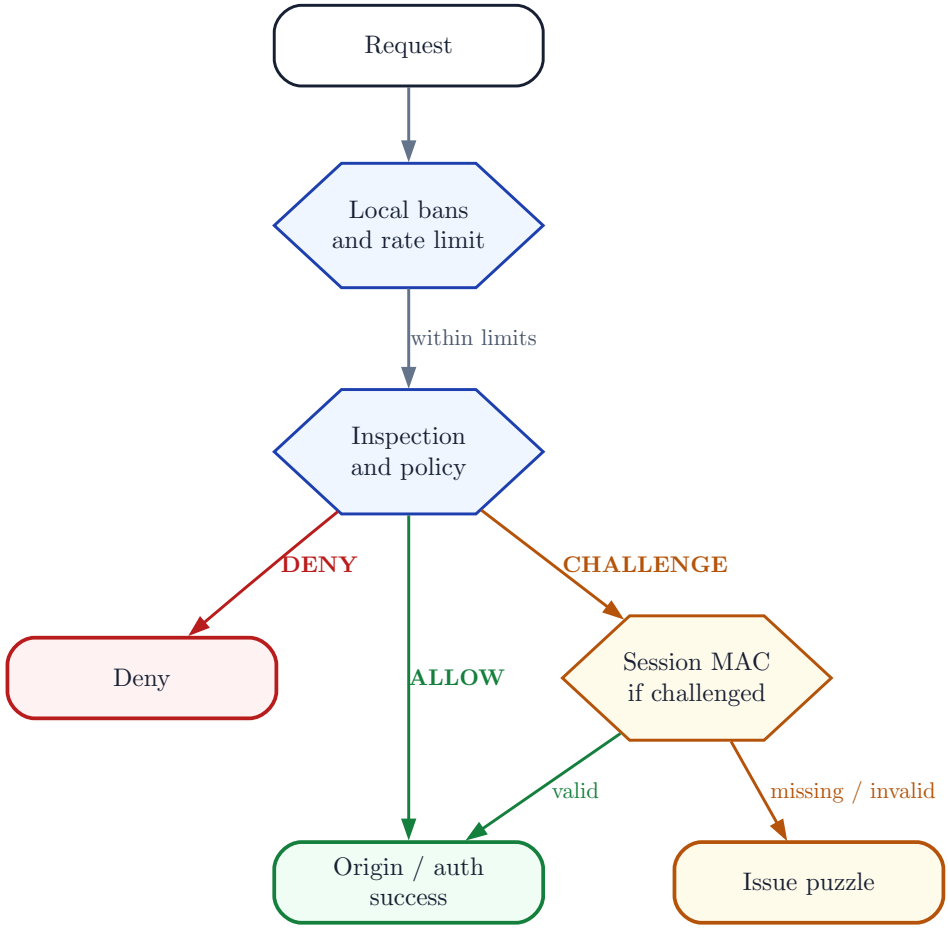
हैं। ग्लोब का *View events* एक्शन चुने गए नोड को बनाए रखता है और पिछले एक घंटे के लिए देश की रखी गई घटनाएँ खोलता है। Statistics पेज का *Security* दृश्य लाइव rate-limit, चैलेंज और प्रतिबंध दरों को एक घंटे, एक दिन, सात दिन या तीस दिन के रखे गए निरीक्षण और हनीपॉट निष्कर्षों के साथ जोड़ता है। श्रेणी, स्रोत और पाथ के लिंक उसी नोड और एक निश्चित समय सीमा के साथ सामान्य घटना वर्कफ़्लो खोलते हैं। घटना फ़िल्टर लागू करने से नई अवधि शुरू होती है। निष्कर्षों में ऑडिट रिकॉर्ड शामिल हैं और वे ब्लॉक किए गए अनुरोधों के योग से अलग हैं; अनुपस्थित प्रतिष्ठा और rule-hit attribution को "not recorded" दिखाया जाता है। इवेंट और ऑडिट फ़िल्टर अपने नियंत्रणों के साथ लेबल रखते हैं, Apply एक्शन को अलग संरेखित करते हैं और अपने कॉलम उपलब्ध सामग्री-चौड़ाई के अनुसार ढालते हैं। एक प्रमाणित WebSocket नेविगेशन के बाद भी बना रहता है और आँकड़े, घटना सारांश, नोड स्थिति, नीति revision, चैलेंज और ऑडिट सारांश ले जाता है। नए घटना और ऑडिट रिकॉर्ड *Load latest records* के पीछे प्रतीक्षा करते हैं ताकि तालिका पढ़ते समय अपनी जगह बनी रहे। नीति अपडेट किसी खुले ड्राफ़्ट को बदले बिना वर्तमान कमिट किए गए और लागू revision दिखाते हैं। प्रवाह काउंटर लाइव अपडेट होते हैं; चुने गए गैर-डिफ़ॉल्ट चैलेंज टाइमिंग विभाजन को स्पष्ट रूप से ताज़ा करें। ऐतिहासिक क्वेरी, विवरण रीड और परिवर्तन HTTP अनुरोध ही रहते हैं।

पेज के अंशों को बुकमार्क किया जा सकता है; Back और Forward प्रमाणित पेज दोबारा खोलते हैं। साइडबार इस ब्राउज़र में थीम और स्पेसिंग के चुनाव भी याद रखता है, जिसमें System डिफ़ॉल्ट थीम है। SID 0007 Committed है। कंसोल अपने golden fixtures और अपने ब्राउज़र वर्कफ़्लो headless Chromium में पास करता है (`cargo xtask browser console`, SID 0010 मानदंड 5)। इसका end-to-end सूट `cmd/sibuna/tests/console_*.rs` है, जिसके PostgreSQL क्लस्टर परिदृश्य `SIBUNA_TEST_DATABASE_URL` सेट होने पर चलते हैं। SID 0010 दर्ज करता है कि कंसोल कंपाइल होने पर, अक्षम, केवल स्टोरेज, या सक्षम और निष्क्रिय होने पर, 16 उत्पाद वर्कलोड उसके बिना बने डेमन के लगभग 1% के भीतर रहते हैं, जो noise के भीतर है। SID 0007 का औपचारिक पृथक्करण गेट, लोड में आठ सक्रिय डैशबोर्ड के साथ, अभी मापा नहीं गया है (भाग VIII)। पूरे WebAssembly एप्लिकेशन की 768 KiB की असंपीड़ित ऊपरी सीमा है, जिसे डेमन की बिल्ड स्क्रिप्ट बिल्ड विफल करके लागू करती है, और 640 KiB का समीक्षा थ्रेशोल्ड है। 1.0.0-beta.3 में मॉड्यूल का आकार 724,330 बाइट है (gzip में 272,568), जो समीक्षा थ्रेशोल्ड से ऊपर है, और SID 0007 इसे समीक्षा के लिए दर्ज करता है। ये प्रोजेक्ट सीमाएँ कंसोल वर्कफ़्लो के लिए जगह छोड़ती हैं; ये ब्राउज़र लोडिंग और प्रतिक्रियाशीलता के मापन की जगह नहीं लेतीं। मॉड्यूल की linear memory माँग पर 16 MiB की घोषित अधिकतम सीमा तक बढ़ती है।

डिप्लॉयमेंट टोपोलॉजी

> **उद्देश्य** स्वायत्त रिवर्स प्रॉक्सी और forward-auth सत्यापनकर्ता को Nginx या Caddy के पीछे डिप्लॉय करें।

रिवर्स प्रॉक्सी



चित्र 24 Shield सतह पर अनुरोध रूटिंग

```
sibuna --port 80 --upstream-host 127.0.0.1 --upstream-port 3000 \
       --secret-file /etc/sibuna/secret --difficulty 16 --algorithm posw
```

स्वीकृत अनुरोध `X-Forwarded-For`, `X-Forwarded-Proto`, `X-Real-IP`, `X-Sibuna-Status` और `X-Sibuna-Rule` हेडर के साथ ऑरिजिन तक पहुँचते हैं। hop-by-hop हेडर, Sibuna द्वारा स्वयं लिखे जाने वाले फ़ील्ड और वैकल्पिक फ़ॉरवर्डिंग फ़ील्ड (`Forwarded`, `X-Forwarded-Host`, `X-Forwarded-Port`) हटा दिए जाते हैं, तब भी जब कोई क्लाइंट उनके नाम अंडरस्कोर के साथ लिखता है (`X_Forwarded_For`)। `X-Real-IP` वह क्लाइंट है जिसके आधार पर अनुरोध का निर्णय हुआ। `X-Forwarded-For` केवल सॉकेट पीयर है, कभी क्लाइंट द्वारा लिखी गई प्रविष्टि नहीं; जब फ़ॉरवर्ड की गई पहचान पर भरोसा हो, तब यह प्राप्त शृंखला और उसके बाद सॉकेट पीयर है। इसलिए ऑरिजिन को क्लाइंट को `X-Real-IP` से या `X-Forwarded-For` के दाएँ छोर से पढ़ना चाहिए, कभी उसकी सबसे बाईं प्रविष्टि से नहीं। ऑरिजिन को मिलने वाला पाथ वह सामान्य रूप है जिसका नीति ने निर्णय किया (SID 0009); जिस पाथ का सामान्य रूप न हो उसे `400` उत्तर मिलता है।

HTTP/1.1 WebSocket अपग्रेड ऑरिजिन का handshake स्वीकार होने से पहले प्रवेश और नीति जाँचों से गुज़रते हैं। subprotocol और extension की बातचीत end-to-end रहती है। दो थ्रेड बाइट रिले करते हैं, प्रति दिशा एक, हर

एक के पास एक निश्चित बफ़र, पहले से पढ़े गए बाइट और half-close सहित; इसलिए अपग्रेड किया गया कनेक्शन दो थ्रेड रखता है, और जब हर कनेक्शन अपग्रेड हो तो कनेक्शन थ्रेड की संख्या `--max-connections` की दोगुनी तक पहुँच सकती है। अपग्रेड किए गए सॉकेट सामान्य कनेक्शन कोटा और shutdown रजिस्ट्री के भीतर रहते हैं, लेकिन HTTP निष्क्रियता समय-सीमा से स्वतंत्र `--websocket-idle-timeout` (डिफ़ॉल्ट 300 सेकंड) का उपयोग करते हैं। किसी भी दिशा का ट्रैफ़िक उस सीमा को ताज़ा करता है, लेकिन जो पीयर पूरी अवधि तक एक भी बाइट स्वीकार नहीं करता, वह रिले समाप्त कर देता है, दूसरी दिशा चाहे जितनी व्यस्त हो। handshake के बाद के frame WAF निरीक्षण के इनपुट नहीं हैं। HTTPS/WSS, Sibuna के निजी HTTP/1.1 listener के सामने TLS टर्मिनेट करने वाले इनग्रेस का उपयोग करता है; Sibuna स्वयं ब्राउज़र का TLS टर्मिनेट नहीं करता।

इनग्रेस ब्राउज़र के साथ HTTP/2 तय कर सकता है और Sibuna को HTTP/1.1 फ़ॉरवर्ड कर सकता है। Sibuna में नेटिव HTTP/2 समर्थन बाद के अपडेट तक स्थगित है; यह व्यवस्था नेटिव gRPC जैसे एप्लिकेशन के लिए end-to-end HTTP/2 अर्थ (semantics) नहीं देती।

Content-Length वाली अनुरोध बॉडी निश्चित बफ़र से स्ट्रीम होती हैं, बाइट और MIME हेडर को बनाए रखते हुए। multipart अपलोड सीमाएँ (boundaries), दोहराए गए फ़ील्ड नाम, फ़ाइल नाम और भाग के प्रकार बनाए रखते हैं। WAF बॉडी के अधिकतम पहले 8 KiB की जाँच करता है: multipart मेटाडेटा और गैर-फ़ाइल फ़ील्ड टेक्स्ट निरीक्षण के इनपुट रहते हैं, जबकि फ़ाइल पेलोड अपारदर्शी होते हैं। जिस भाग का `filename` खाली है, वह एक सामान्य फ़ील्ड है। पहचाने गए बाइनरी शीर्ष-स्तरीय MIME प्रकार (SVG को छोड़कर इमेज, audio, video, PDF, ZIP, gzip, 7z, protobuf और octet-stream) की बॉडी केवल तभी अपारदर्शी है जब उसका प्रीफ़िक्स भी बाइनरी लगे, यानी जब हर सोलह में कम से कम एक बाइट tab, line feed, vertical tab, form feed और carriage return के अलावा कोई कंट्रोल बाइट हो; अन्यथा उसका निरीक्षण टेक्स्ट के रूप में होता है (SID 0004)। JSON, XML, SVG, URL-encoded फ़ॉर्म और अज्ञात प्रकार टेक्स्ट निरीक्षण बनाए रखते हैं। अस्पष्ट या विकृत MIME मेटाडेटा टेक्स्ट निरीक्षण पर लौट आता है। multipart पार्सिंग उसी प्रीफ़िक्स के भीतर 32 भागों और प्रति भाग 2 KiB हेडर तक सीमित है, और delimiter के बाद RFC 2046 की transport padding स्वीकार करती है।

यह अपलोड की संगतता है, फ़ाइल सत्यापन या malware स्कैनिंग नहीं। बैकएंड को क्लाइंट की Content-Type घोषणा पर भरोसा करने के बजाय स्वीकृत मीडिया प्रकार स्वयं लागू करने चाहिए। निरीक्षण प्रीफ़िक्स के बाद के फ़ील्ड, किसी बड़ी अपलोड फ़ाइल के बाद वाले फ़ील्ड सहित, निरीक्षित नहीं होते। संपीड़ित पेलोड निरीक्षण के लिए डीकंप्रेस नहीं किए जाते। अपलोड फिर भी प्रवेश, पाथ, क्वेरी और हेडर जाँचों से गुज़रते हैं और कनेक्शन की समय-सीमाओं के अधीन रहते हैं। chunked अनुरोध बॉडी निरीक्षण से पहले कनेक्शन बफ़र के भीतर डिकोड होती हैं (SID 0009)। जो बॉडी बफ़र के भीतर समाप्त होती है, वह Content-Length के साथ बैकएंड तक पहुँचती है। लंबी बॉडी को Sibuna दोबारा chunk करता है, प्रति रीड एक chunk, इसलिए बैकएंड कभी क्लाइंट के chunk आकार, extension या trailer नहीं देखता। अकेले CR या LF वाली, आकार के आसपास whitespace वाली, विकृत extension वाली या 4 KiB से लंबी chunk पंक्ति 400 के साथ अस्वीकार होती है। 16 KiB से बड़ा trailer खंड भी इसी तरह अस्वीकार होता है। किसी अन्य coding वाले `Transfer-Encoding` को 501 मिलता है। जो बैकएंड chunked अनुरोध पार्स नहीं कर सकते, उन्हें लगभग 44 KiB से छोटी बॉडी के लिए फिर भी Content-Length मिलता है। जिस ऑरिजिन कनेक्शन ने अनुरोध बॉडी ढोई, उसे उसके एक्सचेंज के बाद बंद कर दिया जाता है और वह कभी पूल में नहीं लौटता। `100-continue` की प्रतीक्षा करने वाले क्लाइंट को बॉडी भेजने से पहले यह स्थानीय रूप से मिल जाता है; असमर्थित अपेक्षाओं को 417 मिलता है। Expect हेडर बैकएंड को फ़ॉरवर्ड करने से पहले ही consume हो जाता है।

इनग्रेस के पीछे Forward Auth

`--mode forward_auth` में डेमन इनग्रेस के subrequest का उत्तर `200` (साथ में ऑडिट हेडर `X-Sibuna-Status`, `X-Sibuna-Rule` और `X-Sibuna-Rule-Hash`), चैलेंज के लिए `401`, अस्वीकृति के लिए `403`, या रेट लिमिट होने पर `429` से देता है। इनग्रेस को क्लाइंट का पता और मूल URL फ़ॉरवर्ड करना होगा; forward-auth मोड डिफ़ॉल्ट रूप से उन पर भरोसा करता है। इस listener को निजी रूप से bind करें ताकि केवल इनग्रेस ही जुड़ सके। `X-Forwarded-Uri` (Caddy) या `X-Original-URI` (Nginx), और `X-Forwarded-Method`, नीति मूल्यांकन के लिए एप्लिकेशन

अनुरोध को बहाल करते हैं; मूल पाथ को सामान्य रूप में रखा जाता है, जैसे अनुरोध पंक्ति के पाथ को। दोहराए गए या परस्पर विरोधी मूल-URL फ़ील्ड, और ऐसा मूल URI जो origin-form नहीं है, `400` के साथ अस्वीकार होते हैं। डेमन के आंतरिक रूट हमेशा वास्तविक अनुरोध URI का उपयोग करते हैं। forward-auth उस बॉडी का निरीक्षण नहीं करता जिसे इनग्रेस छोड़ देता है।

```
map $http_upgrade $sibuna_connection_upgrade {
    default upgrade;
    '' close;
}
server {
    listen 443 ssl;
    location / {
        auth_request /__sibuna_auth;
        auth_request_set $sibuna_auth_status $upstream_status;
        auth_request_set $sibuna_retry_after $upstream_http_retry_after;
        auth_request_set $sibuna_status $upstream_http_x_sibuna_status;
        auth_request_set $sibuna_rule $upstream_http_x_sibuna_rule;
        auth_request_set $sibuna_rule_hash $upstream_http_x_sibuna_rule_hash;
        error_page 401 = @sibuna_challenge;
        error_page 500 = @sibuna_auth_error;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection $sibuna_connection_upgrade;
        proxy_set_header Host $host;
        proxy_set_header X-Sibuna-Status $sibuna_status;
        proxy_set_header X-Sibuna-Rule $sibuna_rule;
        proxy_set_header X-Sibuna-Rule-Hash $sibuna_rule_hash;
        proxy_read_timeout 300s;
        proxy_pass http://127.0.0.1:3000;
    }
    location = /__sibuna_auth {
        internal;
        proxy_pass http://127.0.0.1:8080/;
        proxy_pass_request_body off;
        proxy_set_header Content-Length "";
        proxy_set_header X-Original-URI $request_uri;
        proxy_set_header X-Forwarded-Method $request_method;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header X-Forwarded-Uri "";
        proxy_set_header X-Forwarded-For $remote_addr;
        proxy_set_header User-Agent $http_user_agent;
        proxy_set_header Cookie $http_cookie;
    }
    location @sibuna_challenge {
        rewrite ^ /__sibuna/challenge break;
        proxy_pass http://127.0.0.1:8080;
        proxy_pass_request_body off;
        proxy_set_header Content-Length "";
        proxy_set_header X-Original-URI $request_uri;
        proxy_set_header X-Forwarded-Method $request_method;
        proxy_set_header X-Forwarded-For $remote_addr;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
    location @sibuna_auth_error {
        add_header Retry-After $sibuna_retry_after always;
```

```
        if ($sibuna_auth_status = 429) { return 429; }
        return 503;
    }
    location /__sibuna/ {
        proxy_pass http://127.0.0.1:8080;
        proxy_set_header X-Forwarded-For $remote_addr;
    }
}
```

```
example.com {
    handle /__sibuna/* {
        reverse_proxy localhost:8080
    }
    handle {
        forward_auth localhost:8080 {
            uri /
            header_up X-Forwarded-For {remote_host}
            header_up -X-Original-URI
            copy_headers X-Sibuna-Status X-Sibuna-Rule X-Sibuna-Rule-Hash
        }
        reverse_proxy localhost:3000
    }
}
```

`/__sibuna/*` नामस्थान (इंटरस्टिशियल, challenge, verify, सॉल्वर फ़ाइलें) दोनों विधियों में सीधे डेमन तक पहुँचना चाहिए। nginx का error page मूल URI और मेथड आगे भेजता है ताकि इंटरस्टिशियल वह आवश्यकता ले जाए जो प्राधिकरण ने तय की; उनके बिना भी पेज काम करता है, और जारी करना ब्राउज़र द्वारा रिपोर्ट किए गए URL के मूल्यांकन पर लौट आता है। Caddy मूल URI/मेथड मेटाडेटा स्वयं देता है। परस्पर अनन्य (exclusive) `handle` ब्लॉक सुनिश्चित करते हैं कि challenge और verification रूट forward-auth precheck में प्रवेश न करें। एप्लिकेशन WebSocket के लिए Nginx को ऊपर दिखाए गए स्पष्ट Upgrade/Connection हेडर चाहिए। Nginx का error handler दूसरा प्राधिकरण निर्णय लिए बिना या अपलोड बॉडी consume किए बिना आंतरिक चैलेंज रूट रेंडर करता है। उसका auth मॉड्यूल केवल 2xx, 401 और 403 को सीधे स्वीकार करता है, इसलिए उदाहरण rate-limit वाली auth त्रुटि को Retry-After के साथ वापस 429 में बदलता है; अन्य प्राधिकरण विफलताएँ 503 के साथ बंद रहती हैं। Nginx अपनी 403 बॉडी देता है, जबकि Caddy Sibuna का अस्वीकृति पेज आगे भेजता है। दोनों विधियाँ आने वाले Sibuna ऑडिट हेडर को वास्तविक प्राधिकरण रिस्पॉन्स से बदल देती हैं। सामान्य TLS प्रमाणपत्र और एप्लिकेशन की अपलोड/समय-सीमा सीमाएँ इनग्रेस पर कॉन्फ़िगर करें; उदाहरण रूटिंग का तर्क दिखाते हैं। रिपॉज़िटरी अभी इन विधियों को चलते हुए इनग्रेस पर नहीं परखती; जिन फ़ॉरवर्ड-मेटाडेटा नियमों पर वे निर्भर हैं, वे `sibuna-http` के यूनिट परीक्षणों से ढके हैं।

इनग्रेस से JA4 फ़िंगरप्रिंट

JA4 फ़िंगरप्रिंट संक्षेप में बताता है कि कोई क्लाइंट TLS कनेक्शन कैसे खोलता है। इसकी गणना क्लाइंट के पहले handshake संदेश, ClientHello, से होती है। क्लाइंट की TLS लाइब्रेरी यह संदेश किसी भी HTTP बाइट के भेजे जाने से पहले तय कर देती है, इसलिए कोई क्लाइंट अपने HTTP अनुरोधों के भीतर से अपना फ़िंगरप्रिंट नहीं बदल सकता। ब्राउज़र, लाइब्रेरी और बॉट इसमें भिन्न होते हैं। इससे JA4 `User-Agent` मान से अधिक मज़बूत संकेत बनता है, जिसे कोई भी क्लाइंट अपनी इच्छा से लिख सकता है (SID 0014)। JA4 को FoxIO ने परिभाषित किया था। एक फ़िंगरप्रिंट में 36 अक्षर होते हैं, _ से जुड़े तीन भागों में:

```
t13d1516h2_8daaf6152771_02713d6af862
```

```
^^^^^^^^^^ ^^^^^^^^^^^^ ^^^^^^^^^^^^
a          b            c

a  प्रोटोकॉल (t TCP, q QUIC, d DTLS), TLS संस्करण (13 यानी TLS 1.3),
   SNI के साथ d या उसके बिना i, cipher की संख्या (दो अंक),
   extension की संख्या (दो अंक), ALPN कोड (h2 यानी HTTP/2)
b  क्रमबद्ध cipher suites का 12 अंकों का lowercase हेक्साडेसिमल हैश
c  क्रमबद्ध extensions और signature algorithms का 12 अंकों का
   lowercase हेक्साडेसिमल हैश
```

केवल वही घटक ClientHello देखता है जो TLS टर्मिनेट करता है, इसलिए केवल वही JA4 की गणना कर सकता है। Sibuna TLS टर्मिनेट नहीं करता: वह इनग्रेस के पीछे सादा HTTP/1.1 परोसता है, और इनग्रेस ब्राउज़र के साथ TLS, HTTP/2 और HTTP/3 बोलता है। कई इनग्रेस पहले से JA4 की गणना करते हैं। इसलिए हम फ़िंगरप्रिंट की गणना इनग्रेस को करने देते हैं और वह उसे एक अनुरोध हेडर में भेजता है, और Sibuna उसे वहीं से पढ़ता है।

`--ja4-header <name>` उस हेडर का नाम देता है। यह डिफ़ॉल्ट रूप से बंद है और इसका कोई डिफ़ॉल्ट नाम नहीं है; हम हमेशा वही हेडर नाम देते हैं जिसका इनग्रेस उपयोग करता है। Sibuna नाम को छोटे अक्षरों में संग्रहीत करता है और केस की परवाह किए बिना उसका मिलान करता है। वह उन फ़ील्ड के नाम अस्वीकार करता है जिन्हें वह स्वयं लिखता या हटाता है (`X-Forwarded-*`, `X-Real-IP`, `X-Sibuna-*` और `Forwarded`, `-` की जगह `_` के साथ भी)। विकल्प सेट होने पर स्टार्टअप बैनर `JA4: from cf-ja4 (trusted)` जैसी एक पंक्ति छापता है।

हेडर केवल तभी पढ़ा जाता है जब फ़ॉरवर्ड किए गए फ़ील्ड पर भरोसा हो। यह वही निर्णय है जो `X-Forwarded-For` को नियंत्रित करता है: `--trust-forwarded`, या `--no-trust-forwarded` के बिना forward-auth मोड। अविश्वसनीय पीयर से आया हेडर अनदेखा किया जाता है। भरोसे के साथ, किसी अनुरोध में फ़िंगरप्रिंट तब होता है जब उस नाम का ठीक एक फ़ील्ड मौजूद हो और उसका मान JA4 के रूप में हो। विकृत मान, या दोहराया गया फ़ील्ड, अनुरोध को कोई फ़िंगरप्रिंट नहीं देता। दो काउंटर इसे देखते हैं: `sibuna_ja4_present_total` फ़िंगरप्रिंट वाले अनुरोध गिनता है, और `sibuna_ja4_invalid_total` वे अनुरोध गिनता है जिनका हेडर विकृत या दोहराया गया था। अविश्वसनीय पीयर का अनुरोध किसी भी काउंटर को नहीं बढ़ाता।

रिवर्स-प्रॉक्सी मोड में ऑरिजिन हेडर को केवल तभी देखता है जब Sibuna उस पर भरोसा करता है। भरोसे के बिना, Sibuna ऑरिजिन को भेजे जाने वाले अनुरोध से उस नाम का हर फ़ील्ड हटा देता है, तब भी जब कोई क्लाइंट नाम को `-` की जगह `_` से लिखता है। इसलिए कोई क्लाइंट उस नाम से ऑरिजिन को जाली फ़िंगरप्रिंट नहीं थमा सकता। भरोसे के साथ, फ़ील्ड अपनी सटीक वर्तनी में बिना बदले गुज़रता है, और `_` वाला उपनाम फिर भी हटा दिया जाता है। `--ja4-header` के बिना उस नाम का फ़ील्ड एक सामान्य फ़ील्ड है।

> **हेडर का स्वामी इनग्रेस होना चाहिए** फ़िंगरप्रिंट उतना ही विश्वसनीय है जितना उसे लिखने वाला इनग्रेस। Sibuna इनग्रेस द्वारा गणना किए गए मान और क्लाइंट द्वारा भेजे गए मान में अंतर नहीं कर सकता। इसलिए इनग्रेस को Sibuna तक पहुँचने वाले हर रूट पर क्लाइंट द्वारा दी गई हेडर की हर प्रति को अधिलेखित करना, या हटाना, चाहिए। Sibuna के listener को केवल इनग्रेस से कनेक्शन स्वीकार करने चाहिए, जैसे `X-Forwarded-For` के लिए।

Cloudflare. managed transform "Add bot protection headers" ऑरिजिन को भेजे गए अनुरोध में `cf-bot-score`, `cf-verified-bot` और `cf-ja3-hash` के साथ `cf-ja4` जोड़ता है। इसके लिए Bot Management सहित Enterprise प्लान आवश्यक है। Cloudflare और Sibuna के बीच के किसी इनग्रेस को यह फ़ील्ड बिना बदले आगे भेजना चाहिए। हम Sibuna को इस तरह शुरू करते हैं:

```
sibuna --ja4-header cf-ja4 --trust-forwarded -s /etc/sibuna/secret
```

Sibuna के सामने वाले इनग्रेस को केवल Cloudflare के पतों से कनेक्शन स्वीकार करने चाहिए; अन्यथा कोई क्लाइंट Cloudflare को बायपास करके अपना `cf-ja4` भेज सकता है।

nginx. FoxIO का `ja4-nginx-module` nginx के भीतर JA4 की गणना करता है और उसे `$http_ssl_ja4` के रूप में उपलब्ध कराता है। इसके लिए nginx 1.26 या बाद का संस्करण चाहिए, मॉड्यूल के ClientHello capture पैच के साथ, `--with-http_ssl_module` के साथ बिल्ड किया हुआ। `proxy_set_header` क्लाइंट के उसी नाम वाले फ़ील्ड को बदल देता है, इसलिए Sibuna तक पहुँचने वाला हर location हेडर स्वयं सेट करता है:

```
# रिवर्स प्रॉक्सी: Sibuna --ja4-header x-ja4 --trust-forwarded के साथ चलता है
location / {
    proxy_set_header X-JA4 $http_ssl_ja4;
    proxy_set_header X-Forwarded-For $remote_addr;
    proxy_pass http://127.0.0.1:8080;
}

# Forward auth: Sibuna --mode forward_auth --ja4-header x-ja4 के साथ चलता है
location = /__sibuna_auth {
    internal;
    proxy_pass http://127.0.0.1:8080/;
    proxy_set_header X-JA4 $http_ssl_ja4;
    # ... ऊपर की forward-auth विधि के बाकी फ़ील्ड
}
```

forward-auth विधि में वही `proxy_set_header X-JA4 $http_ssl_ja4;` पंक्ति `@sibuna_challenge` में और `location /__sibuna/` में भी होनी चाहिए। जो location इसे छोड़ देता है वह क्लाइंट का अपना `X-JA4` Sibuna को भेज देता है, और Sibuna उस पर भरोसा करता है।

Apache Traffic Server. `ja4_fingerprint` प्लगइन `plugin.config` में `ja4_fingerprint.so` पंक्ति से सक्षम होता है। इसके कोई विकल्प नहीं हैं। यह हर TLS कनेक्शन के अनुरोधों में फ़िंगरप्रिंट वाला एक `ja4` फ़ील्ड, और प्रॉक्सी के नाम वाला एक `x-ja4-via` फ़ील्ड जोड़ता है। हम Sibuna को `--ja4-header ja4 --trust-forwarded` के साथ शुरू करते हैं। प्लगइन का दस्तावेज़ यह नहीं बताता कि वह क्लाइंट द्वारा भेजे गए `ja4` फ़ील्ड को बदलता है या नहीं। मान पर भरोसा करने से पहले हमें अपने संस्करण के लिए यह पता करना होगा, या प्लगइन के अपना फ़ील्ड जोड़ने से पहले क्लाइंट का फ़ील्ड हटाना होगा।

कोई नियम `ja4` फ़ील्ड से फ़िंगरप्रिंट का मिलान करता है (भाग X, नीति स्कीमा)। मान या तो पूरा फ़िंगरप्रिंट होता है, जो बराबर होना चाहिए, या ऐसा प्रीफ़िक्स जिसके बाद `*` हो, और फ़िंगरप्रिंट उसी से शुरू होना चाहिए। अकेला `"*"` किसी भी फ़िंगरप्रिंट से मेल खाता है। फ़िंगरप्रिंट रहित अनुरोध कभी किसी `ja4` नियम से मेल नहीं खाता। लोडर ऐसे पैटर्न को, जिससे कोई फ़िंगरप्रिंट मेल नहीं खा सकता, जैसे बड़े अक्षरों में `T13D*`, `INVALID POLICY JA4` के साथ अस्वीकार करता है। मिलान बाइट की तुलना करता है और कुछ भी आवंटित नहीं करता।

```
{ "rules": [
    { "name": "deny-known-scanner", "action": "DENY",
      "ja4": "t13i010400_0f2cb44170f4_5c4c70b73fa0" },
    { "name": "weigh-no-sni", "ja4": "t13i*", "action": "WEIGH", "weight": 15 },
    { "name": "weigh-headless", "user_agent": "Headless", "action": "WEIGH",
      "weight": 30 },
    { "name": "challenge-tls12-checkout", "path": "/checkout/*", "ja4": "t12*",
      "action": "CHALLENGE" }
] }
```

पहला नियम एक फ़िंगरप्रिंट को ठीक-ठीक deny करता है; मान एक उदाहरण है, कोई ज्ञात स्कैनर नहीं। अगले दो नियम निर्णय लेने के बजाय अनुरोध के स्कोर में भार जोड़ते हैं: ऐसा TLS 1.3 क्लाइंट जिसने कोई सर्वर नाम नहीं भेजा, और

अपने User-Agent से पहचाना गया headless ब्राउज़र। जब कोई बाद वाला नियम पहले निर्णय नहीं लेता, तो दोनों संकेत दिखाने वाले अनुरोध का स्कोर 45 होता है और वह 40 के डिफ़ॉल्ट deny थ्रेशोल्ड तक पहुँच जाता है, जबकि एक संकेत दिखाने वाले अनुरोध को चैलेंज मिलता है। अंतिम नियम checkout पाथ पर उन सभी क्लाइंट को प्रीफ़िक्स से चैलेंज करता है जिन्होंने TCP पर TLS 1.2 तय किया। वह deny के बजाय चैलेंज करता है, क्योंकि कई वैध पुराने क्लाइंट अब भी TLS 1.2 का उपयोग करते हैं। नियमों को JA4 को इसी तरह अन्य संकेतों के साथ जोड़ना चाहिए। JA4 मान किसी TLS लाइब्रेरी और उसके कॉन्फ़िगरेशन का नाम बताता है, किसी व्यक्ति या डिवाइस का नहीं। कई क्लाइंट एक ही लाइब्रेरी साझा करते हैं, और विशेष लाइब्रेरी किसी ब्राउज़र के ClientHello की नकल कर सकती हैं। JA4 स्वचालन की लागत बढ़ाता है; यह पहचान सिद्ध नहीं करता। इसी कारण Sibuna प्रतिबंध और प्रतिष्ठा को पते से कुंजीबद्ध रखता है, फ़िंगरप्रिंट से कभी नहीं।

घटनाएँ (WAF अस्वीकृतियाँ, हनीपॉट हिट और ऑडिट निष्कर्ष) अपने अनुरोध का फ़िंगरप्रिंट, यदि उसमें था, security_incidents.ja4 कॉलम में दर्ज करती हैं। कंसोल इसे घटना के साक्ष्य में "JA4 fingerprint" के रूप में दिखाता है, मान के साथ या "Not recorded"। Events फ़िल्टर फ़िंगरप्रिंट या प्रीफ़िक्स स्वीकार करते हैं। managed-rule संपादक में एक JA4 फ़ील्ड है, जो policies.ja4_pattern में संग्रहीत होता है, और नीति परीक्षण फ़ॉर्म एक फ़िंगरप्रिंट लेता है।

घोषणात्मक नीति

> **उद्देश्य** नियमों, भार, थ्रेशोल्ड, IP नियमों और WAF स्विच के साथ एक नीति फ़ाइल लिखें।

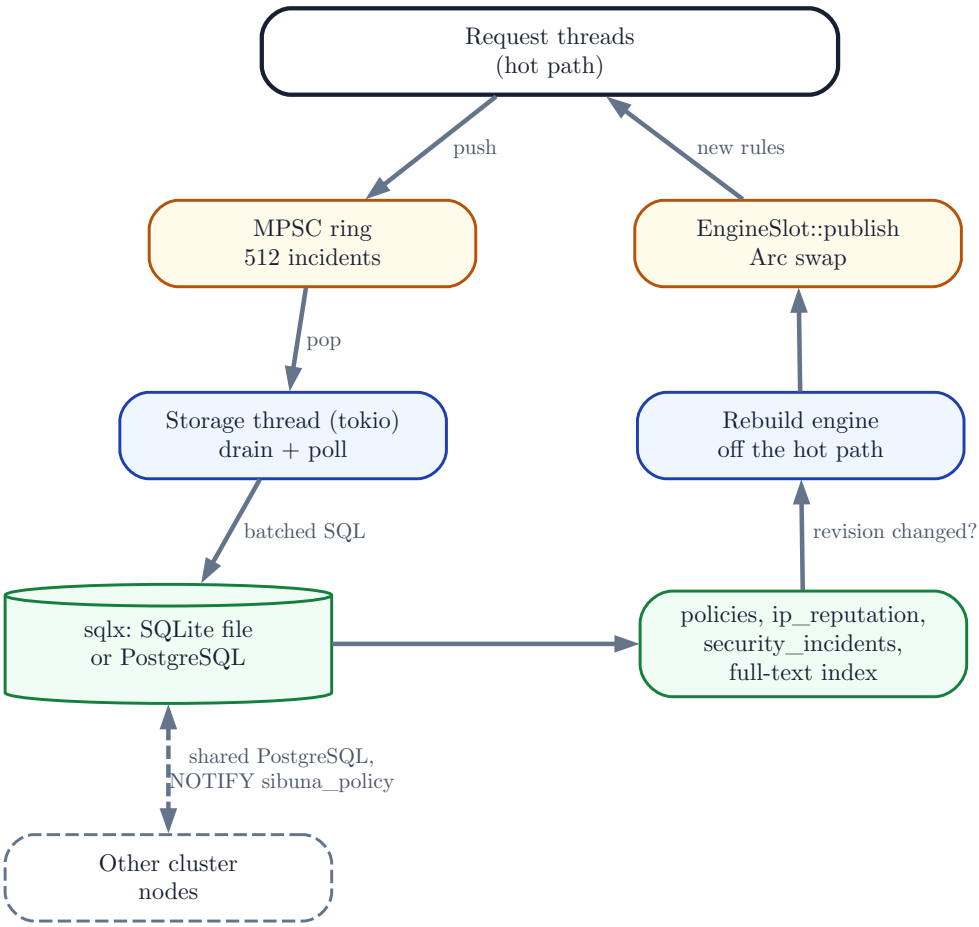
```
{
  "default_action": "CHALLENGE",
  "waf": true,
  "thresholds": { "challenge_at": 10, "deny_at": 40, "bits_step": 5 },
  "ip_rules": { "10.0.0.0/8": "ALLOW", "192.0.2.0/24": "DENY", "2001:db8::/32":
"DENY" },
  "rules": [
    { "name": "deny-cf-workers", "headers": { "CF-Worker": ".*" }, "action":
"DENY" },
    { "name": "deny-amazonbot", "user_agent": "Amazonbot", "action": "DENY" },
    { "name": "api-with-key", "path": "/api/*", "headers": { "X-Api-Key": ".*" },
    "action": "ALLOW" },
    { "name": "protect-checkout", "path": "/checkout/*", "action": "CHALLENGE",
      "challenge": { "difficulty": 20, "algorithm": "posw" } },
    { "name": "headless", "user_agent": "Headless", "action": "WEIGH", "weight":
30 },
    { "name": "internal-vpc", "remote_addresses": ["10.0.0.0/8", "fd00::/8"],
    "action": "ALLOW" }
  ]
}
```

लोडर बंद होकर विफल होता है (fails closed): कोई अज्ञात कुंजी, गलत वर्तनी वाला एक्शन, विकृत पता या सीमा से बाहर की संख्या पूरी फ़ाइल को अस्वीकार कर देती है, निदान नियम और फ़ील्ड का नाम देता है, और डेमन शुरू नहीं होता। अज्ञात विकल्प या सीमा से बाहर के मान वाली कमांड लाइन भी इसी तरह अस्वीकार होती है। मौजूद होने पर rules बिल्ट-इन टेबल की जगह लेता है; ip_rules प्रतिष्ठा trie को भरता है, जो हज़ारों प्रीफ़िक्स तक बढ़ सकता है; waf: false फ़ाइल से Gate सतह चुनता है। दो और कुंजियाँ स्कीमा का भाग हैं (भाग X): inspection हर WAF श्रेणी को disabled, audit या enforce पर सेट करता है, और किसी नियम का limits ऑब्जेक्ट उसे अपना कोटा देता है। किसी नियम का ja4 फ़ील्ड उस JA4 फ़िंगरप्रिंट से मेल खाता है जिसे किसी विश्वसनीय इनग्रेस ने फ़ॉरवर्ड किया, ठीक-ठीक, या ऐसे

प्रीफ़िक्स से जिसके बाद * हो; इसके लिए `--ja4-header` आवश्यक है (देखें "इनग्रेस से JA4 फ़िंगरप्रिंट")। पाथ पैटर्न सामान्य रूप वाले अनुरोध पाथ से मेल खाते हैं (SID 0003), इसलिए पैटर्न उसी रूप में लिखा जाना चाहिए।

स्थायी स्टोरेज और क्लस्टरिंग

उद्देश्य SQLite या PostgreSQL स्टोरेज सक्षम करें, SQL से एक डायनामिक नीति और एक प्रतिबंध जोड़ें, घटनाओं की फ़ॉरेंसिक क्वेरी करें, कैंपेन क्लस्टरिंग समझें, और एक PostgreSQL डेटाबेस पर कई नोड चलाएँ।



चित्र 25 स्टोरेज आर्किटेक्चर: वर्कर कभी डेटाबेस पर नहीं रुकते

`--data-dir /var/lib/sibuna` SQLite फ़ाइल `/var/lib/sibuna/sibuna.db` खोलता है (WAL मोड, `synchronous=NORMAL`, एक busy timeout) और स्टोरेज थ्रेड शुरू करता है। `--database-url` (या `--database-url-file`) इसके बजाय एक PostgreSQL डेटाबेस से जुड़ता है। स्टोरेज थ्रेड एक single-threaded tokio रनटाइम और एक छोटे कनेक्शन पूल का स्वामी है; यही एकमात्र स्थान है जो SQL चलाता है। शुरू होने पर, जब भी नीति revision बदलता है (`policies`, `ip_reputation` या `policy_inspection` में राइट), और जब कोई प्रतिष्ठा प्रविष्टि समाप्त होती है, थ्रेड नीति फ़ाइल और डेटाबेस से इंजन दोबारा बनाता है और उसे प्रकाशित करता है; अनुरोध कभी SQL की प्रतीक्षा

नहीं करते। दोबारा बनाना लोड करने से पहले और बाद में revision पढ़ता है, और यदि बीच में किसी writer ने कमिट किया हो तो अपना उम्मीदवार छोड़ देता है।

स्कीमा

तालिका 25 मुख्य टेबल

टेबल	काम और कुंजी
`policy_revision`	नीति revision, एक पंक्ति; तीनों नीति टेबल पर trigger इसे आगे बढ़ाते हैं।
`policies`	क्रमबद्ध डायनामिक नियम, `id` से कुंजीबद्ध। nullable `ja4_pattern` किसी नियम का JA4 मानदंड रखता है, अधिकतम 37 बाइट।
`policy_inspection`	प्रति श्रेणी संग्रहीत WAF मोड, एक पंक्ति।
`ip_reputation`	स्कोर, हिट की गिनती और समाप्ति, पते या प्रीफ़िक्स से कुंजीबद्ध।
`incident_receipts`	हर नोड की स्थायी घटना रसीद (उसके अंतिम कमिट किए गए बैच के बाद का अगला क्रमांक), नोड से कुंजीबद्ध।
`security_incidents`	अपनी 64-घटक एम्बेडिंग (256 बाइट) के साथ एक सीमित इवेंट रिकॉर्ड, `node_id << 40 | sequence` से कुंजीबद्ध। nullable `ja4` अनुरोध का JA4 फ़िंगरप्रिंट रखता है, ठीक 36 बाइट।
`incidents_fts`	SQLite पर: पाथ और पेलोड के canonical टोकन पर एक FTS5 टेबल। PostgreSQL पर वही टोकन `security_incidents` का `tsvector` कॉलम `search` हैं, GIN इंडेक्स के साथ।
`sibuna_schema`	हर लागू migration, घटक और संस्करण से कुंजीबद्ध।

मुख्य स्कीमा `crates/storage/src/schema.rs` है, प्रति dialect एक बार लिखा गया। कंसोल के साथ घटक `console` 33 टेबल जोड़ता है जिनके नाम `console_` से शुरू होते हैं (SID 0013; `crates/console-storage/src/schema/`)। migration प्रति घटक क्रमांकित हैं और बाइनरी में कंपाइल होते हैं। कोर migration 2 दो JA4 कॉलम जोड़ता है; उससे पहले बना डेटाबेस शुरू होने पर migrate किया जाता है। सभी लंबित migration एक ट्रांज़ैक्शन में चलते हैं, और नोड उस डेटाबेस को अस्वीकार करता है जिसमें किसी घटक का संस्करण उसकी जानकारी से नया हो। छापा गया संक्षिप्त स्कीमा अपग्रेड स्क्रिप्ट नहीं है। SQLite टेबल `STRICT` हैं, और `CHECK` constraint Rust प्रकारों का प्रतिबिंब हैं, इसलिए डेटाबेस ऐसी पंक्ति अस्वीकार करता है जिसे डेमन व्यक्त नहीं कर सकता। गणनीय (enumerated) मान छोटे अक्षरों में संग्रहीत होते हैं: `'DENY'` का सीधा राइट अस्वीकार होता है।

इसका संचालन

`sibuna.db` पर `sqlite3` shell का उपयोग करें (WAL मोड डेमन के चलते समय इसे पढ़ने देता है), या क्लस्टर डेटाबेस पर `psql` का:

```
-- एक डायनामिक नियम जोड़ें; हर नोड अपने अगले tick पर इंजन दोबारा बनाता है।
INSERT INTO policies (id, name, priority, path_pattern, action, difficulty,
algorithm,
  header_matchers, cidr_matchers, weight, enabled, created_at, updated_at)
VALUES ('p-checkout', 'protect-checkout', 10, '/checkout/*', 'challenge', 20,
'posw',
  '{"X-Api": "v2"}', '["10.0.0.0/8"]', 0, 1, unixepoch(), unixepoch());
```

```
-- किसी पते पर पूरे क्लस्टर में एक दिन का प्रतिबंध लगाएँ।
INSERT INTO ip_reputation (ip_or_cidr, reputation_score, banned_until,
  trigger_rule, hits, last_seen)
VALUES ('198.51.100.7', -100, unixepoch() + 86400, 'analyst', 1, unixepoch());

-- फ़ॉरेंसिक: दर्ज पेलोड पर full-text खोज।
SELECT s.id, s.client_ip, s.violation_category, s.path, s.campaign_id
FROM incidents_fts f JOIN security_incidents s ON s.id = f.rowid
WHERE incidents_fts MATCH 'union' ORDER BY rank LIMIT 20;
```

ये स्टेटमेंट SQLite के लिए लिखे गए हैं। PostgreSQL पर `enabled` का मान `TRUE` है, समय `unixepoch()` के बजाय `CAST(extract(epoch FROM now()) AS BIGINT)` है, और खोज `security_incidents` पर `WHERE search @@ CAST('union' AS tsquery)` पढ़ती है। `path` और `offending_payload` कच्चे अनुरोध बाइट रखते हैं (`BLOB` या `BYTEA`)। constraint तोड़ने वाला सीधा राइट डेटाबेस अस्वीकार कर देता है और वह कभी किसी नोड तक नहीं पहुँचता। PostgreSQL के अंतर्गत revision trigger `NOTIFY sibuna_policy` भी जारी करता है, इसलिए दूसरे नोड तुरंत tick करते हैं; polling गारंटी बनी रहती है।

-50 या उससे नीचे के स्कोर trie में `deny` प्रीफ़िक्स बनते हैं, 50 या उससे ऊपर के स्कोर `allow` बनते हैं, और `banned_until` प्रतिबंध को सीमित करता है। हनीपॉट हिट अपने आप -100 का रिकॉर्ड डालते हैं, इसलिए साझा PostgreSQL डेटाबेस के साथ एक नोड पर फँसा जाल सभी नोड पर उस पते पर प्रतिबंध लगा देता है।

ट्रांज़ैक्शन रसीद रीट्राई को सुरक्षित बनाती है

स्टोरेज थ्रेड अधिकतम 32 लंबित रिकॉर्ड इकट्ठा करता है और एक SQL ट्रांज़ैक्शन बनाता है। ट्रांज़ैक्शन घटनाएँ डालता है, full-text इंडेक्स अपडेट करता है, और हनीपॉट से हुए प्रतिष्ठा बदलाव लागू करता है। हर जारीकर्ता (नोड) की एक एकदिष्ट रूप से बढ़ने वाली रसीद है, `incident_receipts` में उसकी पंक्ति। डेटा बदलने वाला हर स्टेटमेंट उस रसीद से सुरक्षित है; ट्रांज़ैक्शन उसे सबसे अंत में आगे बढ़ाता है, और कभी पीछे नहीं।

हल किया गया उदाहरण: पावती गायब हो जाती है मान लें जारीकर्ता 2 क्रमांक 101 से 132 तैयार करता है, जो रसीद 133 पर समाप्त होते हैं। डेटाबेस सभी रिकॉर्ड और रसीद 133 कमिट कर देता है, फिर पावती पहुँचने से पहले कनेक्शन विफल हो जाता है। स्टोरेज थ्रेड तैयार बैच (पहचानकर्ता, एम्बेडिंग और कैंपेन) रखता है और उसे रीट्राई करता है। रसीद पहले से 133 है, इसलिए सुरक्षित राइट कुछ नहीं करते: न घटनाएँ दोगुनी होती हैं न हनीपॉट हिट की गिनती। यदि मूल ट्रांज़ैक्शन rollback हुआ था, तो पुरानी रसीद बनी रहती है और रीट्राई सभी राइट एक बार लागू करता है।

विफल कमिट लंबित बैच को मेमोरी में रखता है और `incident_write_failures` बढ़ाता है। tick फिर भी नीति polling का प्रयास करता है। भरी हुई 512-रिकॉर्ड कतार `incidents_dropped` गिनती है; वह डिस्क की प्रतीक्षा में कभी HTTP रिस्पॉन्स को नहीं रोकती। लंबित बैच 32 रिकॉर्ड की जगह जोड़ता है। कमिट से पहले प्रोसेस का अंत कतार के डेटा को खो सकता है: यह एक सीमित असमकालिक फ़ॉरेंसिक पाथ है, कतार में डालते समय टिकाऊ संदेश कतार नहीं।

अभ्यास 9.1. रसीद अपडेट को डेटा ट्रांज़ैक्शन से बाहर ले जाएँ। क्रैश का एक ऐसा क्रम बनाएँ जो प्रतिष्ठा प्रभाव को दोगुना करे और दूसरा ऐसा जो एक घटना खो दे।

कैंपेन क्लस्टरिंग

हर घटना पेलोड को बाइट trigram (अंक और case एक रूप में मोड़े गए) पर hashing trick से 64-आयामी इकाई सदिश के रूप में एम्बेड किया जाता है और `security_incidents.embedding` में 256 बाइट के रूप में संग्रहीत किया

जाता है। स्टोरेज श्रेड सभी नोड की सबसे हाल की 16,384 एम्बेडिंग मेमोरी में (4 MiB) रखता है, जो हर tick पर डेटाबेस से ताज़ा होती हैं। प्रविष्टि से पहले वह उनमें से सबसे निकट वाली खोजता है, उसी बैच के पहले के रिकॉर्ड सहित; यदि कोसाइन दूरी अधिकतम 0.35 हो तो नई घटना उस घटना के `campaign_id` में शामिल होती है, अन्यथा वह अपने ही पहचानकर्ता के नाम से एक कैंपेन शुरू करती है। विंडो से पुरानी घटनाओं से तुलना नहीं होती। समानता एक ह्यूरिस्टिक है: रिग्रेशन उदाहरण संबंधित SQL पेलोड को समूहित करते हैं, लेकिन यह गारंटी नहीं कि आक्रमणों का हर जोड़ा एक कैंपेन साझा करता है। कोई मॉडल नहीं चलता और कुछ भी प्रशिक्षित नहीं होता।

क्लस्टरिंग

क्लस्टर उन नोड का समुच्चय है जो एक PostgreSQL डेटाबेस साझा करते हैं। `postgres-tls` feature के साथ बिल्ड करें (`cargo build --release -p sibuna --features postgres-tls`; इसमें `postgres` शामिल है) और हर सदस्य को डेटाबेस और उसके अपने नोड id के साथ शुरू करें:

```
sibuna --database-url-file /etc/sibuna/database-url --cluster-node 1 \
       --secret-file /etc/sibuna/secret
```

हर सदस्य को एक ही मास्टर सीक्रेट और 1 से 8,388,607 तक एक अद्वितीय `--cluster-node` का उपयोग करना होगा। URL फ़ाइल में एक `postgres://` URL होता है और वह केवल उसके स्वामी द्वारा पठनीय होनी चाहिए; बैनर कभी URL नहीं छापता। राइट PostgreSQL primary पर जाते हैं। हर नोड का स्टोरेज श्रेड कमिट हुए revision के बदलाव को देखता है (`NOTIFY` से या polling से) और अपना इंजन दोबारा बनाता है।

नोड नेटवर्क के पार साझा डेटाबेस तक पहुँचते हैं, इसलिए हम उस कनेक्शन को एन्क्रिप्ट और सत्यापित करते हैं। हम सर्वर को अपनी certificate authority से हस्ताक्षरित प्रमाणपत्र देते हैं, और URL में उस authority का नाम देते हैं: `postgres://sibuna:...@db.internal:5432/sibuna?sslmode=verify-full&sslrootcert=/etc/sibuna/db-ca.pem`। सर्वर पर हम TCP केवल `pg_hba.conf` की `hostssl` पंक्तियों के ज़रिए स्वीकार करते हैं।

स्टार्टअप पर डेमन डेटाबेस कनेक्शन के बारे में एक पंक्ति छापता है। `verify-full` के साथ वह बताता है कि प्रमाणपत्र और होस्ट नाम सत्यापित हैं। किसी भी कमज़ोर मोड के साथ वह एक चेतावनी और एक संकेत छापता है। `sslmode` के बिना मोड `prefer` होता है: कनेक्शन केवल तभी एन्क्रिप्ट होता है जब सर्वर TLS दे, और सर्वर सत्यापित नहीं होता। `postgres-tls` के बिना बना डेमन `require`, `verify-ca` और `verify-full` को अस्वीकार करता है और अनुपस्थित feature का नाम देता है; वह कभी plaintext पर नहीं लौटता। अस्वीकृत प्रमाणपत्र स्टार्टअप को तुरंत रोक देता है; संकेत हमसे `sslrootcert` जाँचने को कहता है, और यह भी कि URL का होस्ट सर्वर के प्रमाणपत्र में एक नाम हो। `sslrootcert` हमारी authority को सिस्टम trust store में जोड़ता है; वह सिस्टम की authorities को नहीं हटाता (SID 0011, SID 0012)।

Sibuna अपना कोई डेटाबेस रेप्लिकेट नहीं करता और कोई leader नहीं चुनता। रेप्लिकेशन, failover और बैकअप PostgreSQL के ज़िम्मे हैं (streaming replication, Patroni या कोई managed सेवा)। कंसोल के नोड दृश्य में कोई leader नहीं है: वह SQLite नोड के लिए `single` भूमिका और PostgreSQL नोड के लिए `unknown` दिखाता है, और `quorum` का अर्थ केवल इतना है कि नोड का पिछला डेटाबेस round trip सफल रहा। यदि डेटाबेस तक पहुँच न हो, तो 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 श्रेड backoff के साथ रीट्राई करता है; नोड दोबारा जुड़ने तक अपनी अंतिम नीति रखता है। Zaxonlite डेटा डायरेक्टरी (SID 0005) पढ़ी नहीं जाती; नीतियाँ `sibuna console policies export` और `sibuna console policies import` से स्थानांतरित होती हैं, और फ़ॉरेंसिक इतिहास माइग्रेट नहीं होता।

क्लस्टर में चैलेंज रूटिंग

चैलेंज कुंजियाँ `--cluster-node` से बँधी हैं; चैलेंज जारी करना और सत्यापन एक ही सदस्य पर रखें। टोकन साझा सीड का उपयोग करते हैं और दूसरे सदस्यों पर काम करते हैं। स्थानीय रेट कोटा वैश्विक कोटा नहीं बनते, और प्रोसेस रीस्टार्ट spent set को साफ़ कर देते हैं। एक स्थानीय PostgreSQL पर तीन नोड के साथ, सीधे लिखी गई नीति पंक्ति एक

tick के भीतर तीनों तक पहुँचती है, एक नोड पर हनीपॉट प्रतिबंध दूसरों तक पहुँचता है, और घटना पहचानकर्ता कभी नहीं टकराते (`crates/storage/tests/multi_node.rs`, SID 0011 मानदंड 2; यह `SIBUNA_TEST_DATABASE_URL` सेट होने पर चलता है)।

पैकेजिंग

> **उद्देश्य** एक रिलीज़ पैकेज इंस्टॉल और सत्यापित करें, Sibuna को सेवा के रूप में, कंटेनर में या Kubernetes के अधीन चलाएँ, और बाइनरी स्वयं बिल्ड करें।

GitHub पर हर रिलीज़ हर प्लेटफ़ॉर्म के लिए वही बाइनरी देती है, जो रिलीज़ टैग से डिफ़ॉल्ट features के साथ बिल्ड होती है: SQLite स्टोरेज और कंसोल। PostgreSQL क्लस्टर के लिए हमें अपना बिल्ड चाहिए (नीचे देखें)।

प्लेटफ़ॉर्म	रिलीज़ फ़ाइलें
Linux x86-64, ARM64	`sibuna-linux-amd64.tar.gz` और `sibuna-linux-arm64.tar.gz` में एक स्टैटिक musl बाइनरी है जो किसी भी डिस्ट्रीब्यूशन पर चलती है। `sibuna_<version>-1_amd64.deb`, `sibuna_<version>-1_arm64.deb`, `sibuna-<version>-1.x86_64.rpm` और `sibuna-<version>-1.aarch64.rpm` उसे एक systemd सेवा के साथ इंस्टॉल करते हैं; Arch Linux पर `sibuna-bin-<version>-1-x86_64.pkg.tar.zst` यही करता है।
macOS arm64, x86-64	`sibuna-macos-arm64.tar.gz`, `sibuna-macos-amd64.tar.gz`, macOS 12 या बाद के संस्करण के लिए।
Windows x86-64	`sibuna-windows-amd64.zip` में `sibuna.exe` है, Windows 10 या Server 2016 या बाद के संस्करण के लिए।
FreeBSD 15.1, OpenBSD 7.9 (amd64)	`sibuna-<version>-freebsd-15.1-amd64.pkg` और `sibuna-<version>.tgz`।
Homebrew, Kubernetes, कंटेनर	`sibuna.rb`, `helm-sibuna-<version>.tgz`, और amd64 तथा arm64 के लिए इमेज `ghcr.io/insanai/sibuna:<version>`, जिसका डाइजेस्ट `IMAGE-DIGEST.txt` में है।
स्रोत कोड	`sibuna-<version>-source.tar.gz`।

हर आर्काइव और पैकेज में `LICENSE`, `NOTICE`, `LICENSES/` (`THIRD-PARTY-RUST.txt` सहित, जिसमें बाइनरी के क्रेट दर्ज हैं), `README.md`, `SECURITY.md` और `sibuna.build.json` होते हैं। मैनिफ़ेस्ट संस्करण, commit, Rust टारगेट, कंपाइलर संस्करण, features और बाइनरी तथा उसके दो WebAssembly मॉड्यूल के SHA-256 दर्ज करता है। रिलीज़ वर्कफ़्लो कुछ भी प्रकाशित करने से पहले हर बाइनरी को उसके अपने प्लेटफ़ॉर्म पर शुरू करता है, और Debian, Fedora, Arch Linux, FreeBSD और OpenBSD पर पैकेजों को इंस्टॉल करता है, दोबारा इंस्टॉल करता है और हटाता है।

रिलीज़ फ़ाइलें हस्ताक्षरित नहीं हैं। किसी डाउनलोड को इंस्टॉल करने से पहले हम उसे उसी रिलीज़ की `SHA256SUMS` से जाँचते हैं:

```
sha256sum --ignore-missing -c SHA256SUMS          # Linux
shasum -a 256 --ignore-missing -c SHA256SUMS      # macOS, FreeBSD
sha256 -C SHA256SUMS sibuna-1.0.0-beta.3.tgz      # OpenBSD
```

macOS पर ब्राउज़र से डाउनलोड की गई फ़ाइल पर एक क्वारंटीन एट्रिब्यूट लगा होता है जो अहस्ताक्षरित बाइनरी को रोक देता है; जाँच के बाद `xattr -d com.apple.quarantine sibuna` उसे हटा देता है।

systemd सेवा

Debian, RPM और Arch पैकेज `/usr/bin/sibuna`, systemd यूनिट `sibuna.service`, खाता `sibuna` (`systemd-sysusers` के ज़रिए), सहायक स्क्रिप्ट `/usr/lib/sibuna/seed.sh` और सेटिंग्स फ़ाइल `/etc/sibuna/service.env` इंस्टॉल करते हैं। इंस्टॉलेशन कुछ भी शुरू नहीं करता। हम सेटिंग्स फ़ाइल में ऑरिजिन, एक IP पते का लिटरल, सेट करते हैं और सेवा शुरू करते हैं:

```
sudoedit /etc/sibuna/service.env      # SIBUNA_UPSTREAM_HOST, SIBUNA_UPSTREAM_PORT
sudo systemctl enable --now sibuna
curl -s http://127.0.0.1:8080/__sibuna/health
```

सेवा `127.0.0.1:8080` पर सुनती है, जब तक `SIBUNA_HOST` और `SIBUNA_PORT` कुछ और न कहें। `SIBUNA_OPTIONS` आगे के विकल्प देता है, जो रिक्त स्थानों पर विभाजित होते हैं, उदाहरण के लिए `--mode forward_auth` या `--console 127.0.0.1:19446`। हर शुरुआत से पहले, यदि प्रवेश सीड `/var/lib/sibuna/admission.seed` मौजूद न हो, तो `seed.sh` उसे बनाता है (64 हेक्साडेसिमल अंक, mode 0600, स्वामी `sibuna`); वह किसी मौजूदा सीड को कभी नहीं बदलता, क्योंकि नया सीड हर सत्र समाप्त कर देगा। डेमन अपना SQLite डेटाबेस `/var/lib/sibuna` में रखता है, जो mode 0700 की डायरेक्टरी है। यूनिट बिना capabilities के, केवल-पढ़ने वाले सिस्टम (`ProtectSystem=strict`), निजी `/tmp` और डिवाइस, और बिना नए विशेषाधिकारों के चलती है।

अपग्रेड सेटिंग्स फ़ाइल, सीड और संग्रहीत अवस्था को बनाए रखता है, और सेवा को रीस्टार्ट नहीं करता; हम बैकअप के बाद उसे रीस्टार्ट करते हैं। हटाने पर सेवा रुक जाती है और अक्षम हो जाती है, लेकिन खाता, सीड, सेटिंग्स और संग्रहीत अवस्था बने रहते हैं, जिन्हें होस्ट को सेवानिवृत्त करते समय हम स्वयं मिटाते हैं।

FreeBSD और OpenBSD पैकेज केवल `/usr/local/bin/sibuna` और उसके दस्तावेज़ इंस्टॉल करते हैं (`pkg add`, या OpenBSD mirror के बाहर से आए पैकेज के लिए `pkg_add -D unsigned`)। वे कोई सेवा, खाता या संग्रहीत अवस्था नहीं बनाते: हम डेमन को बिना विशेषाधिकार वाले उपयोगकर्ता के रूप में, निजी सीड और डेटा डायरेक्टरी के साथ चलाते हैं।

Homebrew

`sibuna.rb` मशीन के लिए रिलीज़ आर्काइव इंस्टॉल करता है: macOS या Linux, arm64 या x86-64। यह स्रोत से बिल्ड नहीं करता, क्योंकि Homebrew के Rust में `wasm32-unknown-unknown` के लिए मानक लाइब्रेरी नहीं है, जिसकी ज़रूरत दो एम्बेडेड WebAssembly मॉड्यूल के बिल्ड को होती है; आर्काइव में वही बाइनरी हैं जिन्हें रिलीज़ वर्कफ़्लो पहले ही जाँच चुका है। Homebrew फ़ॉर्मूले केवल tap से इंस्टॉल करता है, इसलिए हम फ़ॉर्मूले को एक स्थानीय tap में रखते हैं:

```
brew tap-new "$USER/local"
cp sibuna.rb "$(brew --repository)/Library/Taps/$USER/homebrew-local/Formula/"
brew install "$USER/local/sibuna"
```

फ़ॉर्मूला कोई सेवा शुरू नहीं करता। उसकी caveats दिखाती हैं कि सीड कैसे बनाएँ; उसका परीक्षण डेमन को एक ही डेटा डायरेक्टरी पर दो बार शुरू करता है।

कंटेनर और Kubernetes

इमेज `ghcr.io/insanai/sibuna:<version>` में distroless base पर स्टैटिक बाइनरी है: CA प्रमाणपत्र, उपयोगकर्ता 65532, कोई shell नहीं और कोई पैकेज मैनेजर नहीं। entry point `sibuna` है, इसलिए आर्ग्युमेंट डेमन के विकल्प होते हैं। रिलीज़ वर्कफ़्लो उसे केवल-पढ़ने वाले रूप में, बिना capabilities के, केवल-पढ़ने के लिए माउंट किए गए सीड और संग्रहीत अवस्था के लिए एक वॉल्यूम के साथ चलाता है, और एक बार रीस्टार्ट करता है:

```
docker run -d --read-only --cap-drop=ALL --security-opt=no-new-privileges \
  -v "$PWD/admission.seed:/run/admission.seed:ro" -v sibuna-state:/var/lib/sibuna
\
  -p 8080:8080 ghcr.io/insanai/sibuna:1.0.0-beta.3 \
  --host 0.0.0.0 --secret-file /run/admission.seed --data-dir /var/lib/sibuna/data
\
  --upstream-host 10.0.0.5 --upstream-port 3000
```

Helm चार्ट `helm-sibuna-<version>.tgz` उस इमेज का एक Pod चलाता है। उसे एक Secret चाहिए जिसे हम बनाते हैं (`secret.existingSecret`, कुंजी `admission.seed`), इसलिए कोई सीड कभी values या templates में नहीं आता। Pod, Restricted pod security standard के अधीन, केवल-पढ़ने वाले root फ़ाइल सिस्टम के साथ UID 65532 के रूप में चलता है। `Recreate` अपग्रेड वाली एक रेप्लिका ReadWriteOnce वॉल्यूम पर एक ही लेखक रखती है, जहाँ Sibuna अपनी SQLite अवस्था `data/` में रखता है; अनइंस्टॉल करने पर claim बना रहता है। प्रोब `/__sibuna/health` को कॉल करते हैं। Service ClusterIP है, चार्ट कोई इनग्रेस और कोई कंसोल नहीं बनाता, और उसकी NetworkPolicy सारा ट्रैफ़िक तब तक अस्वीकार करती है जब तक `networkPolicy.ingressFrom` और `networkPolicy.egressTo` क्लाइंट और ऑरिजिन को अनुमति न दें। `upstream.host` एक IP पते का लिटरल होना चाहिए, और `extraArgs` डेमन के आगे के विकल्प देता है। वर्कफ़्लो चार्ट को एक ऐसे kind क्लस्टर में इंस्टॉल, अपग्रेड और अनइंस्टॉल करता है जो Restricted standard लागू करता है।

```
kubectl -n sibuna create secret generic sibuna-admission --from-
file=admission.seed
helm install gateway ./helm-sibuna-1.0.0-beta.3.tgz --namespace sibuna \
  --set secret.existingSecret=sibuna-admission \
  --set upstream.host=10.0.0.5 --set upstream.port=3000
```

स्रोत से बिल्ड करना

हम Rust 1.100 के साथ बिल्ड और परीक्षण करते हैं; न्यूनतम समर्थित संस्करण अभी भी Rust 1.94 है।

हम `rustup toolchain install 1.100.0 --profile minimal --target wasm32-unknown-unknown` से रिलीज़ टूलचेन स्थापित करते हैं और नीचे दिए गए कमांड में `cargo +1.100.0` से उसे चुनते हैं।

- `cargo build --release -p sibuna` डिफ़ॉल्ट features के साथ `target/release/sibuna` बनाता है: SQLite स्टोरेज और कंसोल। sqlx SQLite को साथ में बाँधता (bundle करता) है, और `ring` (rustls) को बिल्ड के समय C कंपाइलर चाहिए; बाइनरी सिस्टम C रनटाइम को डायनामिक रूप से लिंक करती है। बिल्ड स्क्रिप्ट WebAssembly सॉल्वर और कंसोल इंटरफ़ेस भी बनाती है, जिसके लिए `rustup target add wasm32-unknown-unknown` चाहिए; `SIBUNA_POW_WASM=<path>` और `SIBUNA_CONSOLE_WASM=<path>` इसके बजाय पहले से बने मॉड्यूल एम्बेड करते हैं।
- `cargo build --release -p sibuna --no-default-features` बिना किसी बाहरी क्रेट के मेमोरी में चलने वाला फ़ायरवॉल बनाता है; तब `--data-dir` और `--database-url` शुरू होते समय `StorageDisabled` के साथ अस्वीकार होते हैं, और `sibuna console` `CONSOLEBUILD` के साथ।

- `cargo build --release -p sibuna --features postgres-tls` डेटाबेस तक सत्यापित TLS के साथ PostgreSQL क्लस्टर जोड़ता है। इसे OpenSSL की ज़रूरत नहीं।
- रिलीज़ की स्टैटिक Linux बाइनरी `cargo build --release --locked -p sibuna --target x86_64-unknown-linux-musl` (या `aarch64-unknown-linux-musl`) से बनती हैं, जिसमें C कंपाइलर `musl-gcc` होता है (`CC_x86_64_unknown_linux_musl=musl-gcc`)।

रिलीज़ प्रोफ़ाइल fat LTO और एक codegen unit का उपयोग करती है। वर्कफ़्लो दोनों WebAssembly मॉड्यूल एक बार बिल्ड करता है और उन्हें हर प्लेटफ़ॉर्म को देता है, इसलिए किसी रिलीज़ की सभी बाइनरी बाइट-दर-बाइट समान मॉड्यूल एम्बेड करते हैं। किसी निर्भरता के बदलने के बाद `cargo xtask licenses` फ़ाइल `LICENSES/THIRD-PARTY-RUST.txt` को फिर से बनाता है।

Windows पर डेमन Ctrl+C, Ctrl+Break और कंसोल बंद होने के इवेंट पर रुकता है; वह Windows सेवा के रूप में नहीं चलता। Windows फ़ाइलों में mode bits नहीं होते, इसलिए Sibuna ऐसी कुंजी, क्रेडेंशियल या डेटाबेस URL फ़ाइल को अस्वीकार नहीं कर सकता जिसे दूसरे पढ़ सकते हों: हम उन फ़ाइलों को ACL से सीमित करते हैं। कंसोल के मेमोरी और CPU गेज `/proc` से पढ़े जाते हैं और दूसरे सिस्टम पर उपलब्ध नहीं हैं।

अभ्यास 9.2. वह नीति फ़ाइल और `ip_reputation` पंक्तियाँ लिखें जिनसे एक staging नेटवर्क (`10.20.0.0/16`) चैलेंज को बायपास करे, `/admin/*` अनुक्रमिक कार्य के 20 वर्क बिट्स माँगे, और `X-Partner-Key` से पहचाने जाने वाले एक पार्टनर scraper को प्रति 10 सेकंड 30 अनुरोधों पर प्रवेश मिले।
संकेत: डेमन के फ़्लैग वह प्रति-क्लाइंट सीमा तय करते हैं जिससे हर अनुरोध पहले मिलता है; कोई नियम `limits` में अपना कोटा रख सकता है।

अपरिवर्तनीय नियम समझाएँ. किसी ऑपरेटर को स्टोरेज थ्रेड और इंजन स्लॉट के संदर्भ में समझाएँ कि `policies` में एक पंक्ति जोड़ना बिना रीस्टार्ट के और बिना किसी अनुरोध को प्रतीक्षा कराए प्रभावी क्यों होता है।

भाग X

डेस्क संदर्भ और निदान

एंडपॉइंट, मेट्रिक, त्रुटि सूची, नीति स्कीमा और स्टोरेज टेबल, सब एक स्थान पर।

एंडपॉइंट

उद्देश्य `/__sibuna/` के अंतर्गत हर रूट जानें, और यह भी कि हर एक क्या लौटाता है।

तालिका 27 आरक्षित रूट

एंडपॉइंट	मेथड	कार्य
`/__sibuna/challenge`	GET	इंटरस्टिशियल पेज
`/__sibuna/challenge.json?need=&path=`	GET	एक स्टेटलेस चैलेंज जारी करता है: चैलेंज वाले रिस्पॉन्स का आवश्यकता टिकट उसे तय करता है, अन्यथा रिपोर्ट किए गए URL का सामान्य रूप में मूल्यांकन होता है (8 KiB से ऊपर `414`, सामान्य रूप न रखने वाले पाथ के लिए `400`)। `{id, algorithm, difficulty, challenges, expires_at}` लौटाता है
`/__sibuna/verify`	POST	`{"challenge_id", "nonce"}` या `{"challenge_id", "proof"}` स्वीकार करता है; `Set-Cookie` के साथ `200`, निदान के साथ `400`, या कनेक्शन बफ़र से बड़ी बॉडी के लिए `413`
`/__sibuna/wasm/sibuna-pow.wasm`	GET	सॉल्वर मॉड्यूल, 14,744 बाइट (`benchmarks/results/primitives-latest.json` में `wasm_bytes`), एक घंटे तक cache योग्य
`/__sibuna/worker.js`	GET	WASM और JavaScript प्रूवर वाला Web Worker, एक घंटे तक cache योग्य
`/__sibuna/honeypot`	GET	कॉल करने वाले पर `--ban-seconds` के लिए प्रतिबंध लगाता है और एक घटना दर्ज करता है; `403`। केवल शीर्ष-स्तर का अनुरोध प्रतिबंध लगवाता है (`Sec-Fetch-Dest` न हो या `document` हो, और cross-site `Sec-Fetch-Site` न हो); किसी अन्य को `404` उत्तर मिलता है और उसे गिना जाता है
`/__sibuna/health`	GET	इंजन के नाम, संस्करण, प्रॉक्सी मोड और प्रमाण एल्गोरिदम के साथ JSON liveness स्टेटस
`/__sibuna/metrics`	GET	Prometheus टेक्स्ट प्रारूप (संस्करण 0.0.4)

`/__sibuna/` के अंतर्गत किसी भी अन्य पाथ को `404` उत्तर मिलता है। जारी करना और सत्यापन अलग `--challenge-rate-limit` बजट खर्च करते हैं, और प्रतिबंधित पते को हर रूट पर `403` से अस्वीकार किया जाता है। चैलेंज रिस्पॉन्स `X-Sibuna-Status: CHALLENGE` ले जाते हैं; स्वीकृत अनुरोध अपस्ट्रीम की ओर `X-Sibuna-Status: PASS` और `X-Sibuna-Rule` ले जाते हैं (और forward-auth उत्तरों में हेक्साडेसिमल में `X-Sibuna-Rule-Hash`)। जो क्लाइंट HTML स्वीकार नहीं करता, उसे `{"error":"challenge_required","challenge":"/__sibuna/challenge.json?need=<ticket>"}` के साथ `401` मिलता है।
अलग, वैकल्पिक प्रबंधन listener `/console/` देता है। इसके एडमिनिस्ट्रेटर को `sibuna init-admin` से स्थानीय रूप से प्रारंभ करें, फिर पहले साइन-इन पर अस्थायी पासवर्ड बदलें। `/console/api/stats` और `/console/ws` के लिए प्रमाणित सत्र चाहिए (kiosk सत्र सहित)। WebSocket सीमित स्नैपशॉट, delta और gap recovery के साथ आँकड़े, इवेंट, नोड, नीति, चैलेंज और ऑडिट को मल्टीप्लेक्स करता है। प्रबंधन पीयर `/console/peer` से जुड़ते हैं, और हर API रूट `/console/api/` के अंतर्गत है। forward-auth मोड में ऑरिजिन त्रुटि काउंटर नहीं देखे जाते। भाग IX कंसोल के वर्कफ़्लो, HTTPS डिप्लॉयमेंट और SID 0007 की वर्तमान सीमाओं का दस्तावेज़ देता है।

मेट्रिक

Prometheus टेक्स्ट प्रारूप में `sibuna_<name>_total` के रूप में दिखाए जाने वाले काउंटर, इस क्रम में:

तालिका 28 काउंटर

काउंटर	कब बढ़ता है
`incidents_persisted`, `incident_batches`	वे रिकॉर्ड और ट्रांज़ैक्शन जिनके कमिट की पुष्टि स्टोरेज थ्रेड ने की
`incidents_dropped`, `incident_write_failures`	रिंग भरी होने के कारण अस्वीकार किए गए कतार push, और कमिट के विफल प्रयास
`requests`	पार्स किया गया अनुरोध डिस्पैच तक पहुँचा (आरक्षित रूट सहित)
`allowed`, `denied`, `challenged`	नीति निर्णय लिया गया (स्वीकृत, अस्वीकृत, चैलेंज जारी या दोबारा जारी)
`challenges_issued`	`/__sibuna/challenge.json` ने एक चैलेंज रिकॉर्ड जारी किया
`solutions_accepted`, `solutions_rejected`	`/__sibuna/verify` ने किसी प्रमाण को स्वीकार या अस्वीकार किया
`rate_limited`	GCRA ने किसी अनुरोध को अस्वीकार किया (`429`): अनुरोध सीमा, चैलेंज बजट या किसी नियम का कोटा
`banned`	किसी प्रतिबंधित पते को अस्वीकार किया गया, या हनीपॉट ने किसी पर प्रतिबंध लगाया
`bans_issued`	एक प्रतिबंध दर्ज किया गया (हनीपॉट या किसी नियम कोटा का प्रतिबंध)
`proxied`, `upstream_errors`	एक अनुरोध ऑरिजिन तक रिले किया गया, या ऑरिजिन विफल हुआ (`502`)
`parse_errors`	विकृत हेड, दोहराया गया `Host`, सामान्य रूप न रखने वाला लक्ष्य (`400`), या असमर्थित transfer coding (`501`) अस्वीकार किया गया
`overloaded`	`--max-connections` या `--max-connections-per-ip` से आगे के कनेक्शन को `503` उत्तर मिला

काउंटर	कब बढ़ता है
`bans_refused`	प्रतिबंध दर्ज नहीं हुआ, क्योंकि उसकी कुंजी जो भी स्लॉट ले सकती थी उन सभी में एक प्रभावी प्रतिबंध था
`forwarded_invalid`	विश्वसनीय फ़ॉरवर्ड किया गया क्लाइंट पता पार्स नहीं हुआ; अनुरोध का निर्णय सॉकेट पीयर से हुआ
`honeypot_refused`	हनीपॉट हिट से किसी पर प्रतिबंध नहीं लगा, क्योंकि उसे किसी व्यक्ति ने नहीं, एक पेज ने माँगा था
`rate_evictions`	लिमिटर में किसी आगमन ने किसी दूसरे सक्रिय क्लाइंट की सेल को हटा दिया
`ja4_present`	`--ja4-header` और विश्वसनीय फ़ॉरवर्डिंग के साथ, अनुरोध में उस नाम का ठीक एक हेडर JA4 फ़िंगरप्रिंट के साथ था
`ja4_invalid`	`--ja4-header` और विश्वसनीय फ़ॉरवर्डिंग के साथ, हेडर विकृत या दोहराया गया था; अनुरोध का मूल्यांकन बिना फ़िंगरप्रिंट के हुआ

एक रीट्राई रिकॉर्ड खोए बिना `incident_write_failures` बढ़ा सकता है, क्योंकि लंबित बैच रखा जाता है। किसी अंतराल में हुई बढ़ोतरी की निगरानी करें; केवल कुल योग कतार की गहराई नहीं बताते।

स्टेटस कोड

तालिका 29 डेटा plane के स्टेटस कोड

कोड	कब
200	स्वीकृत (forward-auth), इंटरस्टिशियल (चैलेंज की ज़रूरत वाला HTML नेविगेशन), आंतरिक रूट
302	वर्तमान प्रोटोकॉल में उपयोग नहीं होता; सत्यापन `200` उत्तर देता है और पेज दोबारा लोड होता है
400	विकृत अनुरोध, दोहराया गया `Host` फ़ील्ड, सामान्य रूप न रखने वाला लक्ष्य, अमान्य फ़ॉरवर्ड किया गया मेटाडेटा, विकृत बॉडी, या Elm-शैली के निदान के साथ अस्वीकृत समाधान
401	HTML स्वीकार न करने वाले क्लाइंट के लिए, या forward-auth मोड में, चैलेंज आवश्यक
403	नीति या WAF द्वारा अस्वीकृति, प्रतिबंधित पता, हनीपॉट
404	`/__sibuna/` के अंतर्गत अज्ञात पाथ, या ऐसा हनीपॉट हिट जो शीर्ष-स्तर का अनुरोध नहीं है
413	समाधान की बॉडी 64 KB कनेक्शन बफ़र से बड़ी
414	रिपोर्ट किया गया चैलेंज URL 8 KiB से बड़ा
417	असमर्थित अनुरोध अपेक्षा (expectation); `100-continue` स्थानीय रूप से संभाला जाता है
429	GCRA सीमा पार; `Retry-After` सेकंड में
431	अनुरोध हेड 16 KB से बड़ा
501	chunked के अलावा किसी अन्य coding वाला `Transfer-Encoding`
502	ऑरिजिन तक पहुँच नहीं, या उसका रिस्पॉन्स हेड विकृत था या 16 KB से बड़ा था

कोड	कब
503	कनेक्शन सीमा (`--max-connections` या `--max-connections-per-ip`) पूरी, या नोड ड्रेन हो रहा है; उत्तर के बाद सॉकेट बंद कर दिया जाता है

त्रुटि सूची

`INVALID COMMAND LINE` हर उस विकल्प के लिए स्टार्टअप रोक देता है जिसे डेमन नहीं पहचानता, अपनी दस्तावेज़ित सीमा से बाहर के हर मान के लिए, और अमान्य, अनुपस्थित या दोहराए गए मोड चयन के लिए। ब्लॉक विकल्प, दिया गया मान, अपेक्षित सीमा और त्रुटि (`UnknownOption`, `MissingValue`, `InvalidValue`, `InvalidMode`, `MissingMode`, `DuplicateMode`, Zaxonlite क्लस्टर विकल्प के लिए `RemovedOption`, एक-दूसरे को बाहर करने वाले दो स्टोरेज विकल्पों के लिए `ConflictingOptions`, `--cluster-node` के बिना डेटाबेस URL के लिए `MissingOption`) का नाम देता है, और एक संकेत के साथ समाप्त होता है। डेटाबेस URL `[redacted]` के रूप में दिखाया जाता है। `--mode reverse_proxy` या `--mode forward_auth` एक बार दें; `-m` समतुल्य छोटा विकल्प है। अस्वीकृत नीति फ़ाइल अपनी त्रुटि के शीर्षक वाला एक ब्लॉक छापती है (उदाहरण के लिए `INVALID POLICY ACTION`), जिसमें फ़ाइल, त्रुटि का नाम, नियम, फ़ील्ड और मान होते हैं। ऐसा `ja4` पैटर्न जिससे कोई फ़िंगरप्रिंट मेल नहीं खा सकता, `INVALID POLICY JA4` (`InvalidJa4`) छापता है; ऐसे पैटर्न वाला कंसोल प्रबंधन दस्तावेज़ उसी त्रुटि नाम के साथ अस्वीकार होता है। कंसोल विकल्पों की जाँच अलग से होती है: `CONSOLE002` विकृत या दोहराए गए कंसोल विकल्प का नाम देता है, और `CONSOLE001` किसी नियम को तोड़ने वाले कंसोल कॉन्फ़िगरेशन का (स्टोरेज के बिना `StorageRequired`)।

> **इम्प्लीमेंटेशन: `ErrorKind::explain`** हर डोमेन त्रुटि को एक सीमा पंक्ति, एक व्याख्या और एक `Hint:` से जोड़ता है। `crates/core/src/errors.rs` में (क्रेट `sibuna-core`)।

तालिका 30 चैलेंज और सत्र की त्रुटियाँ

त्रुटि	कारण	संकेत
`MalformedChallenge`	पहचानकर्ता एक सही रूप वाला चैलेंज रिकॉर्ड नहीं है	नया चैलेंज लाएँ और उसे बिना बदले सबमिट करें
`InvalidChallengeTag`	टैग प्रमाणित नहीं होता; इस क्लस्टर ने जारी नहीं किया या संपादित किया गया	चैलेंज जाली नहीं बनाए जा सकते; नया माँगें
`ChallengeExpired`	चैलेंज TTL से पुराना, या भविष्य में जारी किया गया	नया चैलेंज माँगें
`FingerprintMismatch`	किसी दूसरे पते या User-Agent से सबमिट किया गया	उसी क्लाइंट से सबमिट करें जिसने उसे मँगाया था
`DifficultyNotMet`	Hashcash नॉनस में आवश्यक शून्य बिट नहीं हैं	नॉनस खोजते रहें
`InvalidProof`	अनुक्रमिक कार्य का प्रमाण कमिट किए गए लेबल नहीं खोलता	जारी की गई गहराई और ओपनिंग के लिए प्रूवर को पूरा चलाएँ
`WrongSolutionType`	PoSW चैलेंज के लिए नॉनस या Hashcash के लिए प्रमाण	समाधान के फ़ील्ड को एल्गोरिदम से मिलाएँ

त्रुटि	कारण	संकेत
`DoubleSpendAttempt`	चैलेंज पहले ही खर्च हो चुका था	चैलेंज एक बार ही उपयोग होते हैं
`StoreFull`	spent set भरा हुआ है	चैलेंज TTL घटाएँ या `--spent-tags` बढ़ाएँ
`InvalidTokenSignature`	कुकी का टैग या हस्ताक्षर विफल	इंटरस्टिशियल के ज़रिए दोबारा प्रमाणित हों
`TokenExpired`	कुकी अपनी समाप्ति से आगे	दोबारा प्रमाणित हों
`TokenBoundAddressMismatch`	कुकी किसी दूसरी क्लाइंट पहचान से प्रस्तुत की गई	कुकी साझा नहीं की जा सकतीं

कंसोल का API विफलता का उत्तर `{"error":"<CODE>","hint":"<sentence>"}` से देता है; hint वह टेक्स्ट है जो इंटरफ़ेस दिखाता है (`crates/console/src/http/reply.rs`)।

तालिका 31 ऑपरेटर से संबंधित कंसोल विफलता कोड

कोड	स्टेटस	अर्थ
`CONSOLE002`	400	अनुरोध विकृत है, बहुत बड़ा है, या अपने रूप में अनुमत नहीं है
`CONSOLE003`	503 / 429	स्टोरेज जॉब कतार भरी है (32 जॉब), या पासवर्ड सत्यापनकर्ता व्यस्त है
`CONSOLEQUORUM`	503	डेटाबेस ने अनुरोध की 10 सेकंड की प्रतीक्षा के भीतर उत्तर नहीं दिया; परिणाम अज्ञात है, और डेटा plane अंतिम लागू नीति से चलता रहता है
`CONSOLE004`	503	सेवा उत्तर नहीं दे सकती: इस बिल्ड में उपलब्ध नहीं, रुक रही है, या आंतरिक विफलता
`CONSOLENODE`	400 / 409 / 503	नोड कमांड अस्वीकार हुई: वह किसी दूसरे नोड का नाम देती है (400), पुराना boot या control revision (409), या वह तब आई जब कोई दूसरी कमांड चल रही थी या किसी पूर्णता को दर्ज नहीं किया जा सका (503); रीट्राई से पहले operation receipt की जाँच करें
`CONSOLEMUTATION`	429	एक सत्र से एक मिनट में साठ से अधिक प्रबंधन परिवर्तन (mutation)

नीति स्कीमा

```
{
  "default_action": "ALLOW" | "DENY" | "CHALLENGE",
  "waf": true | false,
  "inspection": { "path_traversal" | "sqli" | "xss" | "rce": "disabled" | "audit"
| "enforce" },
  "thresholds": { "challenge_at": int, "deny_at": int, "bits_step": int },
  "ip_rules": { "<cidr>": "ALLOW" | "DENY" | "CHALLENGE", ... },
  "rules": [                                                    // अधिकतम 128
    {
```

```
      "name": "<kebab-case>",                                    // 1 से 128
printable बाइट
      "path" | "path_regex": "<pattern>",                        // अधिकतम 512 बाइट
      "user_agent" | "user_agent_regex": "<pattern>",
      "headers" | "headers_regex": { "<Header>": "<pattern>" }, // अधिकतम 4
      "remote_addresses" | "cidrs": ["<cidr>", ...],             // अधिकतम 8, IPv4
या IPv6
      "action": "ALLOW" | "DENY" | "CHALLENGE" | "WEIGH",
      "weight": int,                                             // केवल WEIGH,
-1000..1000
      "challenge": { "difficulty": <work bits>, "algorithm": "hashcash" |
"posw" },
      "limits": { "rate": int, "window_seconds": int, "ban_seconds": int },
      "ja4": "<fingerprint>" | "<prefix>*"                       // 36 बाइट, या
प्रीफ़िक्स और *
    }
  ]
}
```

`challenge_at` और `deny_at` -1000 से 1000 के बीच होते हैं, `deny_at` का मान `challenge_at` के बराबर या उससे ऊपर होता है, और `bits_step` 1 से 1000 के बीच। `challenge` ब्लॉक केवल `CHALLENGE` नियमों का होता है, 1 से 64 वर्क बिट्स के साथ। `limits` किसी नियम को उसका अपना कोटा देता है: `rate` 1 से 1,000,000 और `window_seconds` 1 से 86,400 आवश्यक हैं, `ban_seconds` (अधिकतम 86,400) वैकल्पिक है, और `WEIGH` नियम सीमाएँ नहीं रख सकते। `ja4` या तो एक पूरा JA4 फ़िंगरप्रिंट है, जिसके बराबर फ़िंगरप्रिंट होना चाहिए, या ऐसा प्रीफ़िक्स जिसके बाद `*` हो, और फ़िंगरप्रिंट उसी से शुरू होना चाहिए; अकेला `*` किसी भी फ़िंगरप्रिंट से मेल खाता है, और फ़िंगरप्रिंट रहित अनुरोध किसी `ja4` नियम से मेल नहीं खाता। ऐसा पैटर्न जिससे कोई फ़िंगरप्रिंट मेल नहीं खा सकता, `InvalidJa4` के साथ अस्वीकार होता है (भाग IX, इनग्रेस से JA4 फ़िंगरप्रिंट)। `ip_rules` का प्रतिष्ठा trie अधिकतम 8,192 नोड रखता है। हर नियम हर मानदंड की एक ही वर्तनी का उपयोग कर सकता है।

पैटर्न का व्याकरण: `.*` या `*` किसी भी चीज़ से मेल खाते हैं; `^...$` एक सटीक पाथ को बाँधता है; अंत में `*`, `/*` या `.*` प्रीफ़िक्स है; `/` से शुरू होने वाला पैटर्न सटीक पाथ है; बाकी सब केस-असंवेदी सबस्ट्रिंग है। पाथ पैटर्न सामान्य रूप वाले पाथ से मेल खाते हैं (खंडों के बीच एक स्लैश, कोई dot खंड नहीं, unreserved अक्षर unescaped, अन्य escape बड़े अक्षरों वाले हेक्साडेसिमल अंकों के साथ), इसलिए `;`, `%2F`, `%5C` या `\` रखने वाला पैटर्न किसी से मेल नहीं खाता, और पाथ मिलान केस-संवेदी है (SID 0003)।

स्टोरेज टेबल

`policy_revision`, `policies` (अधिकतम 37 बाइट के nullable `ja4_pattern` मानदंड के साथ), `policy_inspection`, `ip_reputation`, `incident_receipts`, `security_incidents` (256-बाइट एम्बेडिंग और nullable 36-बाइट `ja4` फ़िंगरप्रिंट के साथ), full-text इंडेक्स (SQLite पर `incidents_fts`, canonical टोकन पर FTS5; PostgreSQL पर GIN इंडेक्स के साथ `tsvector` कॉलम `search`) और `sibuna_schema`; कंसोल 33 `console_*` टेबल जोड़ता है। संग्रहीत एक्शन, एल्गोरिदम और मोड के नाम छोटे अक्षरों में हैं। घटना id `node_id << 40 | sequence` हैं, जो बिना समन्वय के पूरे क्लस्टर में अद्वितीय हैं; नोड id 1 से 8,388,607 तक होते हैं।

बिल्ड टारगेट

तालिका 32 बिल्ड और रिपॉज़िटरी कमांड

कमांड	परिणाम
`cargo build --release -p sibuna`	SQLite स्टोरेज और कंसोल वाला डेमन (डिफ़ॉल्ट features)
`cargo build --release -p sibuna --no-default-features`	मेमोरी में चलने वाला डेमन, स्टोरेज या कंसोल के बिना और किसी बाहरी क्रेट के बिना
`cargo build --release -p sibuna --features postgres-tls`	डेमन जो एक साझा PostgreSQL डेटाबेस से भी जुड़ता है, सत्यापित TLS के साथ (केवल `postgres` feature TLS के बिना जुड़ता है)
`cargo test`	यूनिट परीक्षण, चलते हुए डेमन पर integration परीक्षण, स्टोरेज और कंसोल परीक्षण (`SIBUNA_TEST_DATABASE_URL` सेट होने पर PostgreSQL वाले भी)
`cargo clippy --all-targets`, `cargo fmt --check`	लिंट और फ़ॉर्मेटिंग (`max_width = 99`)
`cargo xtask check`	70-पंक्ति फ़ंक्शन, 1,408-पंक्ति फ़ाइल और 99/108-कॉलम पंक्ति की सीमाएँ; कोई टैब नहीं
`cargo xtask browser`	headless Chromium में जाँचें (`sibuna-browser`)
`cargo xtask bench <primitives|product|latency>`	रिलीज़ मोड के बेंचमार्क; परिणाम `benchmarks/results/<command>-latest.json` में
`cargo build -p sibuna-pow-wasm --target wasm32-unknown-unknown --profile wasm`	केवल ब्राउज़र मॉड्यूल (कंसोल इंटरफ़ेस के लिए `-p sibuna-console-ui`)
`cargo run -p sibuna-geoip --bin geoip-snapshot -- ...`	देश डेटा का सत्यापन करें और एम्बेड करने योग्य GeoIP स्नैपशॉट लिखें
`guidedog build` / `guidedog dsc`	यह पुस्तक / SID रिकॉर्ड

डेमन की बिल्ड स्क्रिप्ट (`cmd/sibuna/build.rs`) दोनों WebAssembly मॉड्यूल को `wasm` प्रोफ़ाइल के साथ वर्कस्पेस की टारगेट डायरेक्टरी में बनाती है, और कंसोल इंटरफ़ेस 768 KiB से अधिक होने पर बिल्ड विफल कर देती है। `wasm32-unknown-unknown` टारगेट के बिना बिल्ड के लिए `SIBUNA_POW_WASM` और `SIBUNA_CONSOLE_WASM` इसके बजाय पहले से बने मॉड्यूल का नाम देते हैं। `SIBUNA_GEOIP_DATA` एम्बेड करने के लिए एक GeoIP स्नैपशॉट का नाम देता है; इसे बिल्ड के समय डिकोड किया जाता है, और कंसोल के बिना बिल्ड इसे अस्वीकार करता है (`GEOIP001`)।

अभ्यास 10.1. हल के बीच में Wi-Fi से सेलुलर पर जाने के बाद एक क्लाइंट को `CLIENT FINGERPRINT MISMATCH` शीर्षक के साथ `400` मिलता है। टोकन और चैलेंज के प्रारूपों से कारण समझाएँ, और इंटरस्टिशियल में वह सबसे छोटा बदलाव प्रस्तावित करें जो इससे सहज रूप से उबर जाए।

अपरिवर्तनीय नियम समझाएँ. बिना देखे उन एंडपॉइंट की सूची बनाएँ जिन्हें Sibuna के सामने का रिवर्स प्रॉक्सी ऑरिजिन के बजाय डेमन तक रूट करे, और बताएँ कि हर एक क्यों आवश्यक है।

भाग XI

त्वरित संदर्भ कार्ड

इसे टर्मिनल के पास रखें: बिल्ड करना, चलाना, कॉन्फ़िगर करना, रूट करना, निगरानी और निदान। पूरी तालिकाएँ भाग X में हैं; इस कार्ड में वह है जो ऑपरेटर टाइप करता है।

बिल्ड

```
cargo build --release -p sibuna                          # डेमन + SQLite + कंसोल
cargo build --release -p sibuna --no-default-features    # मेमोरी में, कोई बाहरी क्रेट नहीं
cargo build --release -p sibuna --features postgres-tls  # साझा PostgreSQL क्लस्टर
rustup target add wasm32-unknown-unknown   # या SIBUNA_POW_WASM /
SIBUNA_CONSOLE_WASM दें
cargo test            # यूनिट, चलते डेमन, स्टोरेज और कंसोल के परीक्षण
cargo clippy --all-targets && cargo fmt --check
cargo xtask check     # 70-पंक्ति / 1,408-पंक्ति / 99-कॉलम की जाँच
cargo xtask browser   # headless Chromium में जाँचें
cargo xtask bench product   # wrk के तहत पूरे उत्पाद -> benchmarks/results/
guidedog build        # यह पुस्तक (guidedog dsc: SID रिकॉर्ड)
```

चलाना

```
# Shield (डिफ़ॉल्ट): PoW + WAF, :3000 तक प्रॉक्सी
sibuna -p 8080 -u 3000 -s /etc/sibuna/secret
# Gate: केवल PoW
sibuna --gate -p 8080 -u 3000 -s /etc/sibuna/secret
# Nginx/Caddy के पीछे forward auth (X-Forwarded-For पर भरोसा करता है)
sibuna -m forward_auth --host 127.0.0.1 -p 8080 -s /etc/sibuna/secret
# Edge: <dir>/sibuna.db में स्थायी नीतियाँ, प्रतिष्ठा, फ़ॉरेंसिक
sibuna -D /var/lib/sibuna -s /etc/sibuna/secret
# विश्वसनीय इनग्रेस से JA4 (Cloudflare का हेडर; nginx मॉड्यूल: x-ja4; ATS: ja4)
sibuna --ja4-header cf-ja4 --trust-forwarded -p 8080 -u 3000 -s /etc/sibuna/secret
# क्लस्टर सदस्य: साझा PostgreSQL, अद्वितीय नोड id
sibuna --database-url-file /etc/sibuna/database-url --cluster-node 1 -s /etc/
sibuna/secret
head -c 32 /dev/urandom | xxd -p -c 64 > /etc/sibuna/secret
```

कंसोल (वैकल्पिक प्रीव्यू)

```
# डेमन बंद रहते हुए प्रारंभ करें, फिर कंसोल शुरू करें
sibuna init-admin admin --data-dir ./data
sibuna --data-dir ./data --console 127.0.0.1:19446
```

`http://127.0.0.1:19446/console/` खोलें और अस्थायी पासवर्ड बदलें। नोड को ग्लोब पर रखने के लिए `--console-location 1.3521,103.8198` जोड़ें। दूरस्थ पहुँच के लिए एक HTTPS प्रॉक्सी, एक स्पष्ट ऑरिजिन और trusted-proxy CIDR चाहिए; भाग IX का पालन करें। SID 0007 की स्थिति Committed है। कंसोल headless Chromium में अपने ब्राउज़र वर्कफ़्लो पास करता है (SID 0010, मानदंड 5); उसका औपचारिक प्रदर्शन-पृथक्करण गेट अभी खुला है (भाग VIII)।

सबसे पहले जानने योग्य फ़्लैग

फ़्लैग	डिफ़ॉल्ट	अर्थ
`-d, --difficulty`	16	वर्क बिट्स (PoSW गहराई = बिट्स - 3)
`-a, --algorithm`	posw	`posw` या `hashcash`
`--posw-challenges`	16	प्रति प्रमाण ओपनिंग
`--token-ttl`	86400	सत्र के सेकंड
`--challenge-ttl`	300	चैलेंज के सेकंड
`--token-scheme`	mac	`mac` या `ed25519`
`--rate-limit / --rate-window`	100 / 10	प्रति विंडो GCRA बर्स्ट
`--ban-seconds`	3600	हनीपॉट प्रतिबंध
`-w, --workers`	CPU की संख्या	accept थ्रेड
`--max-connections`	1024	फिर `503`
`--max-connections-per-ip`	64	प्रति पीयर पता (IPv6: प्रति `--ipv6-prefix`), फिर `503`
`--trusted-hops`	1	क्लाइंट = दाईं ओर से n-वीं `X-Forwarded-For` प्रविष्टि
`--ja4-header`	बंद	इनग्रेस के JA4 फ़िंगरप्रिंट वाला हेडर; केवल भरोसा होने पर पढ़ा जाता है
`--idle-timeout`	15	रीपर के सेकंड
`-P, --policy-file`	कोई नहीं	JSON नीति
`--trust-forwarded`	बंद	forward auth में चालू
`--secure-cookie`	बंद	`Secure` जोड़ें

`/__sibuna/` के अंतर्गत एंडपॉइंट

रूट	उत्तर
`challenge`	इंटरस्टिशियल HTML
`challenge.json?need=&path=`	`{id, algorithm, difficulty, challenges, expires_at}`
`verify` (POST)	`{challenge_id, nonce | proof}` -> `200` + `Set-Cookie`, अन्यथा `400`
`wasm/sibuna-pow.wasm`	15,181-बाइट सॉल्वर
`worker.js`	WASM और JS प्रूवर वाला वर्कर
`honeypot`	कॉल करने वाले पर प्रतिबंध, `403`; सबरिसोर्स या cross-site fetch के लिए `404`
`health`	`{"status":"ok",...}`
`metrics`	Prometheus टेक्स्ट

इन सबको डेमन तक रूट करें, कभी ऑरिजिन तक नहीं।

स्टेटस कोड

कोड	कब
200	स्वीकृत (forward auth), इंटरस्टिशियल, आंतरिक रूट
400	विकृत अनुरोध या लक्ष्य, अस्वीकृत समाधान (निदान वाली बॉडी)
401	चैलेंज आवश्यक: गैर-HTML क्लाइंट या forward auth
403	नीति या WAF द्वारा अस्वीकृति, प्रतिबंध, हनीपॉट
404	अज्ञात `/__sibuna/` रूट, अस्वीकृत हनीपॉट हिट
413	समाधान की बॉडी 64 KB बफ़र से बड़ी
414	रिपोर्ट किया गया चैलेंज URL 8 KiB से बड़ा
417	असमर्थित अनुरोध अपेक्षा (expectation)
429	GCRA सीमा, `Retry-After` सेकंड
431	हेड 16 KB से बड़ा
501	chunked के अलावा कोई अन्य transfer coding
502	ऑरिजिन तक पहुँच नहीं या उसका उत्तर विकृत
503	कनेक्शन सीमाएँ पूरी, या नोड ड्रेन हो रहा है

हेडर

- अपस्ट्रीम *(प्रॉक्सी)*: `X-Forwarded-For` (सॉकेट पीयर, विश्वसनीय होने पर प्राप्त शृंखला के बाद), `X-Forwarded-Proto`, `X-Real-IP` (जिस क्लाइंट का निर्णय हुआ), `X-Sibuna-Status: PASS`, `X-Sibuna-Rule: <rule>`; hop-by-hop और वैकल्पिक फ़ॉरवर्डिंग फ़ील्ड हटा दिए जाते हैं, अंडरस्कोर वाली वर्तनियाँ भी।
- *Forward-auth* उत्तर: `X-Sibuna-Status`, `X-Sibuna-Rule`, `X-Sibuna-Rule-Hash`।
- चैलेंज उत्तर: `X-Sibuna-Status: CHALLENGE`।
- कुकी: `__sibuna_token=<75 chars, 139 with ed25519>; Path=/; Max-Age=<ttl>; HttpOnly; SameSite=Lax[; Secure]`

नीति फ़ाइल का ढाँचा

```
{ "default_action": "CHALLENGE", "waf": true,
  "thresholds": {"challenge_at": 10, "deny_at": 40, "bits_step": 5},
  "ip_rules": {"10.0.0.0/8": "ALLOW", "2001:db8::/32": "DENY"},
  "rules": [
    {"name": "api", "path": "/api/*", "headers": {"X-Api-Key": ".*"}, "action":
"ALLOW"},
    {"name": "checkout", "path": "/checkout/*", "action": "CHALLENGE",
     "challenge": {"difficulty": 20, "algorithm": "posw"}},
    {"name": "login", "path": "^/login$", "action": "CHALLENGE",
     "limits": {"rate": 10, "window_seconds": 60, "ban_seconds": 600}},
    {"name": "old-tls", "ja4": "t12*", "action": "CHALLENGE"},
    {"name": "headless", "user_agent": "Headless", "action": "WEIGH", "weight":
30} ] }
```

JA4: पूरा फ़िंगरप्रिंट, या प्रीफ़िक्स और *; फ़िंगरप्रिंट नहीं, तो मेल नहीं। पैटर्न: `*` या `.*` कुछ भी; `^...$` सटीक; अंत में `*` प्रीफ़िक्स; शुरू में `/` सटीक पाथ; अन्यथा केस-असंवेदी सबस्ट्रिंग। पाथ का मिलान सामान्य रूप (normal form) में होता है। क्रम: WAF -> trie deny/allow -> नियम -> स्कोर -> बायपास -> trie challenge -> बॉट -> डिफ़ॉल्ट।

SQL में स्टोरेज (Edge)

```
INSERT INTO policies (id, name, priority, path_pattern, action, difficulty,
algorithm,
  header_matchers, cidr_matchers, weight, enabled, created_at, updated_at)
  VALUES ('p1', 'protect', 10, '/checkout/*', 'challenge', 20, 'posw', '{}', '[]',
0, 1,
  unixepoch(), unixepoch());
INSERT INTO ip_reputation (ip_or_cidr, reputation_score, banned_until,
trigger_rule, hits, last_seen)
  VALUES ('198.51.100.7', -100, unixepoch() + 86400, 'analyst', 1, unixepoch());
SELECT s.client_ip, s.violation_category, s.path, s.campaign_id
  FROM incidents_fts f JOIN security_incidents s ON s.id = f.rowid
  WHERE incidents_fts MATCH 'union' ORDER BY rank LIMIT 20;
```

SQLite (`sqlite3 <dir>/sibuna.db`); एक्शन छोटे अक्षरों में हैं। स्कोर ≤ -50 -> deny प्रीफ़िक्स; ≥ 50 -> allow। क्लस्टर: हर नोड पर एक ही सीक्रेट, एक PostgreSQL डेटाबेस, `--features postgres-tls`, `--`

`database-url-file <file> --cluster-node N` जिसमें N, 1 से 8,388,607 तक अद्वितीय हो; कोई सदस्य सूची, लीडर या क्लस्टर TLS नहीं (SID 0011)।

निदान

शीर्षक	अर्थ
`MALFORMED CHALLENGE`	id संपादित या छोटा कर दिया गया; नया लें
`INVALID CHALLENGE TAG`	इस सीड या नोड ने जारी नहीं किया
`CHALLENGE EXPIRED`	TTL से पुराना या घड़ी का अंतर > 60 s
`CLIENT FINGERPRINT MISMATCH`	पता या User-Agent बदल गया
`DIFFICULTY NOT MET`	नॉनस में शून्य बिट कम हैं
`INVALID PROOF OF SEQUENTIAL WORK`	ओपनिंग रूट तक नहीं पहुँचतीं
`WRONG SOLUTION TYPE`	PoSW के लिए नॉनस या Hashcash के लिए प्रमाण
`DOUBLE SPEND PREVENTED`	चैलेंज पहले ही उपयोग हो चुका
`STORE FULL`	spent set भर गया; TTL घटाएँ या `--spent-tags` बढ़ाएँ
`NETWORK BINDING MISMATCH`	कुकी किसी दूसरे क्लाइंट में कॉपी की गई
`INVALID COMMAND LINE`	अज्ञात विकल्प, गलत मान, या हटाया गया Zaxonlite विकल्प

जिन मेट्रिक पर अलर्ट लगाएँ: `denied`, `rate_limited`, `overloaded`, `upstream_errors`, `incidents_dropped`, `incident_write_failures`, `bans_refused`, `forwarded_invalid` (सभी `sibuna_*_total`)।

याद रखने योग्य संख्याएँ

चैलेंज id 70 अक्षर (36-बाइट पेलोड + 16-बाइट टैग); टोकन 75 अक्षर (40 + 16), Ed25519 के साथ 139; प्रमाण के बाइट $32(1 + t(n + 1))$; PoSW गहराई = बिट्स - 3, सीमा 4 से 24; अनुरोध बफ़र 64 KB; हेड सीमा 16 KB; निरीक्षित बॉडी 8 KB; spent set 1,048,576 टैग (`--spent-tags`); रेट सेल प्रति टेबल 262,144 (`--rate-table-cells`); प्रतिबंध 65,536 स्लॉट (`--ban-table-slots`); प्रति क्लाइंट पता 64 कनेक्शन; घटना रिंग 512; बैच 32; सबसे हाल की 16,384 एम्बेडिंग पर कैंपेन दूरी 0.35; keep-alive प्रति कनेक्शन 4,096 अनुरोध; ऑरिजिन पूल 256 सॉकेट; स्टोरेज जॉब कतार 32।

इनग्रेस रूटिंग

भाग IX, इनग्रेस के पीछे Forward Auth, में दी गई Nginx या Caddy विधि का उपयोग करें। `/__sibuna/*` को सीधे रूट करें; एप्लिकेशन अनुरोधों को मूल URI, मेथड और विश्वसनीय क्लाइंट पते के साथ अधिकृत करें। आने वाले Sibuna ऑडिट हेडर को प्राधिकरण के परिणाम से बदलें। Nginx को चैलेंज, 429 और अनुपलब्ध auth के लिए स्पष्ट हैंडलिंग चाहिए।

`reverse_proxy` स्वीकृत HTTP/1.1 बॉडी और WebSocket ले जाता है; `forward_auth` प्रवेश देता है और उन्हें इनग्रेस ले जाता है। छोड़ी गई auth बॉडी और ऑरिजिन रिस्पॉन्स का Sibuna निरीक्षण या गिनती नहीं कर सकता। TLS

इनग्रेस पर टर्मिनेट होता है; नेटिव HTTP/2 स्थगित है। अपलोड Content-Length या chunked coding का उपयोग करते हैं; निरीक्षण अधिकतम 8 KiB प्रीफ़िक्स देखता है, और फ़ाइल के बाइट अपारदर्शी रहते हैं।

चुनिंदा समाधान

ये समाधान तर्क की जाँच हैं, अभ्यासों का विकल्प नहीं। यदि कोई परिणाम अलग आए, तो हम पहले धारणाओं की तुलना करते हैं: समय अंतरालों के अंतिम बिंदु, किसे कार्य गिना गया है, और कोई राइट कमिट हुआ है या नहीं, ये असहमति के आम स्रोत हैं।

भाग I: लागत और प्रायिकता

1.1. दस लाख प्रयासों को 100 अनुरोधों में बाँटने पर प्रति अनुरोध 10,000 प्रयास होते हैं। यदि कुल अनुरोध संख्या 100 ही रहती है, तो सत्र साझा करने से यह गणित नहीं बदलता। वास्तविक सिस्टम को यह तय करना होता है कि क्या दूसरी प्रोसेस फ़िंगरप्रिंट और रूटिंग की बाइंडिंग को संतुष्ट कर सकती है; ये शर्तें लागत बँटवारे के समीकरण से बाहर हैं।
1.2. अतिरिक्त दर $1000 - 600 = 400$ रिकॉर्ड/s है। 512 रिकॉर्ड की खाली कतार तरल मॉडल में $512/400 = 1.28$ सेकंड दिलाती है। वास्तविक उत्पादक बर्स्ट में आते हैं और उपभोक्ता बैच कमिट करता है, इसलिए पहला रिकॉर्ड ठीक कब छूटेगा, यह आगमन और कमिट के समय पर निर्भर करता है।

भाग II: पूर्व कार्य

2.1. दो बार कंपाइल किया गया एक ही स्रोत स्टेटमेंट, नॉनस एन्कोडिंग, ट्री के आकार या ओपनिंग के लेआउट पर स्वयं से असहमत नहीं हो सकता; विचलन के लिए कंपाइलर में बग होना ज़रूरी होगा। यह इस बारे में कुछ नहीं कहता कि निर्माण (construction) सुदृढ़ है या नहीं (क्रिप्टोग्राफ़िक समीक्षा), ब्राउज़र रनटाइम उसे सही चलाता है या नहीं (इसे WebAssembly और JavaScript प्रूवर के बीच byte-identical तुलना जाँचती है), या पैरामीटर कैलिब्रेट किए गए हैं या नहीं (मापन)।
2.2. (a) Sibuna Gate या Anubis, एक बाइनरी; (b) Sibuna Shield, SafeLine, या Cloudflare (Pro प्लान या उससे ऊपर); (c) साझा PostgreSQL डेटाबेस (`--database-url`) के साथ Sibuna Edge, साझा Valkey स्टोर के साथ Anubis (अवस्था के लिए, प्रतिबंधों के लिए नहीं), या Cloudflare। Docker होस्ट या सब्सक्रिप्शन के बिना तीनों शर्तें पूरी करने वाला एकमात्र उत्पाद Sibuna Shield है, जिसके नोड एक PostgreSQL डेटाबेस साझा करते हैं (`--database-url` और `--cluster-node`); इसका निर्णय "किस रूप में चलता है", "निरीक्षण" और "मल्टी-नोड" पंक्तियाँ करती हैं।

भाग III: क्रिप्टोग्राफ़ी

3.0. टेल का अनुमान $e^{-4.6} \approx 0.010$ है। नॉनस स्पेस को बाँटने से वह दर बदलती है जिस पर प्रयास पूरे होते हैं, हर स्वतंत्र प्रयास की सफलता की प्रायिकता नहीं। दोहराए गए प्रयास या समन्वय का कार्य किसी वास्तविक समानांतर इम्प्लीमेंटेशन को कम कुशल बना सकते हैं।
3.1. यदि हर कमिटमेंट q प्रायिकता से सफल होता है, तो सभी g के विफल होने की प्रायिकता $(1-q)^g$ है। कम से कम एक के सफल होने की प्रायिकता $1-(1-q)^g$ है। यह सूत्र कमिटमेंट बनाने की लागत, क्वेरी बजट, प्रयासों के बीच की निर्भरता, या किसी विशेष स्वीकृति प्रायिकता वाला कमिटमेंट खोजने के कार्य को नहीं गिनता।
3.2. गहराई 17 और 16 ओपनिंग पर प्रमाण में $32(1 + 16 \times 18) = 9248$ बाइट होते हैं। 32 ओपनिंग पर इसमें $32(1 + 32 \times 18) = 18464$ बाइट होते हैं। ओपनिंग के बाइट दोगुने हो जाते हैं; 32-बाइट रूट अब भी एक ही बार भेजा जाता है।
3.3. वर्क लेवल के बिना किसी सस्ते रूट पर अर्जित सत्र किसी महँगे रूट पर ऐसे अनुरोध को प्रवेश दे देता जिसके लिए अधिक कार्य माँगा गया था; `timestamp` के बिना कोई टोकन "भविष्य में" बनाया जा सकता था और अपनी नीति से अधिक

समय तक चल सकता था; `expiry` के बिना वह कभी समाप्त न होता; `rule_hash` के बिना अपस्ट्रीम यह नहीं जान पाता कि किस नियम ने क्लाइंट को प्रवेश दिया; `fingerprint` के बिना किसी दूसरी मशीन पर कॉपी की गई कुकी स्वीकार हो जाती। टैग को पाँचों को ढकना चाहिए, क्योंकि उसके बाहर छोड़ा गया कोई भी फ़ील्ड मनचाहे ढंग से बदला जा सकता था, और सत्यापनकर्ता यह पहचान नहीं पाता।

भाग IV: प्रोटोकॉल

4.1. दोनों ब्राउज़र पता और User-Agent साझा करते हैं, इसलिए वे फ़िंगरप्रिंट, रेट-लिमिट सेल और बैन स्लॉट साझा करते हैं। वे चैलेंज साझा नहीं करते (हर एक अपने नॉनस के साथ अपना रिकॉर्ड लाता है), न spent-set प्रविष्टि, न कुकी। दूसरे ब्राउज़र की कुकी दोबारा उपयोग करने के लिए NAT पर मौजूद हमलावर को उसे दूसरी मशीन से पढ़ना होगा: तब फ़िंगरप्रिंट मेल खा जाएगा, इसीलिए साझा पते पर बाइंडिंग से अधिक `HttpOnly` और `SameSite` मायने रखते हैं।

भाग V: अवस्था और क्रम

5.1. हेड और बॉडी का पहला भाग (64 KB में हेड के बाद जितना समा सके) कनेक्शन बफ़र में होते हैं और `write_all(request.body)` से भेजे जाते हैं। शेष बाइट कभी पूरे के पूरे बफ़र में नहीं आते: `relay_body` उन्हें क्लाइंट रीडर से 16 KB के टुकड़ों में पढ़ता है और हर टुकड़ा ऑरिजिन को लिखता है। हेड पहले जाना चाहिए, क्योंकि ऑरिजिन को बॉडी से पहले `Content-Length` चाहिए, और क्योंकि ऑडिट हेडर हेड का ही भाग हैं।
5.2. उत्सर्जन अंतराल 100 ms है और बर्स्ट टॉलरेंस 9,900 ms है। एक निष्क्रिय क्लाइंट तुरंत 100 अनुरोध भेज सकता है, फिर हर 100 ms पर एक। 3,000 ms के समावेशी अंतिम बिंदु तक सीमा 130 है। एक निश्चित 10-सेकंड काउंटर पहली विंडो में 100 और विंडो बदलते ही 100 और की अनुमति देगा; एक sliding log किसी भी 10 सेकंड में ठीक 100 की अनुमति देगा, लेकिन उसे प्रति क्लाइंट 100 टाइमस्टैम्प रखने होंगे।

भाग VI: एल्गोरिदम

6.1. किसी नाम में एपॉस्ट्रॉफ़ी और `order` नाम का कोई सामान्य पैरामीटर SQL अभिव्यक्ति स्थापित नहीं करते। सबस्ट्रिंग डिटेक्टर दोनों में से किसी को भी सिंटैक्स समझ सकता है। संरचनात्मक डिटेक्टर ऑपरेटर और लिटरल संबंधों सहित अतिरिक्त साक्ष्य खोजता है। यह फिर भी सीमित निरीक्षण ही है, हर एप्लिकेशन भाषा का पार्सर नहीं।
6.2. LDAP फ़िल्टर कोष्ठकों, `|`, `&`, `!` और `=` से बनते हैं। डिटेक्टर को `(` और `)` के लिए एक बाइट क्लास (नई) चाहिए, और वह `|` तथा `&` के लिए मौजूदा `=` और shell-separator क्लास दोबारा उपयोग कर सकता है। उसे `)(` या `(|(` जैसे अनुक्रम पर सक्रिय होना चाहिए, किसी अकेले कोष्ठक पर नहीं, उसी कारण से जिसके लिए SQL डिटेक्टर एक quote या tautology माँगता है।

भाग VII: ब्राउज़र इंजन

7.1. डिफ़ॉल्ट 16 वर्क बिट्स (16 ओपनिंग के साथ गहराई 13) पर WebAssembly के मान लिए गए 20 ms, लैपटॉप पर JavaScript में $50 \times 20 = 1000$ ms, यानी एक सेकंड, और धीमे फ़ोन पर चार सेकंड हो जाते हैं। ये आँकड़े मान्यताएँ हैं; नीति तय करने से पहले हम अपने क्लाइंट के डिवाइस मापते हैं। एक समझदार नीति: पेज को `fallback` संदेश आगे भेजने दें, उसे सर्वर पर मेट्रिक के रूप में लॉग करें, और कठिनाई सबके लिए घटाने के बजाय केवल उस नियम के लिए घटाएँ जो WebAssembly को ब्लॉक करने वाले ज्ञात क्लाइंट से मेल खाता है।

भाग VIII: मूल्यांकन

8.2. Gate की `admitted` पंक्ति के लिए चार में से $11.84 \times 211{,}345/10^6 \approx 2.50$ कोर व्यस्त थे, इसलिए डेमन CPU से बँधा नहीं था और सीमा लोड जनरेटर ने तय की। प्रति अनुरोध 11.84 us CPU सब-माइक्रोसेकंड वर्गीकरण लागत से कहीं ऊपर है, क्योंकि उत्पाद की पंक्ति में कर्नेल का सॉकेट रीड और राइट, पार्सर, सत्र टैग जाँच, मेट्रिक की बढ़ोतरी, रिस्पॉन्स फ़ॉर्मेटिंग, writer flush, और लोड जनरेटर तथा कनेक्शन थ्रेड के बीच शेड्यूलर का हस्तांतरण भी शामिल है। प्रिमिटिव सूट इनमें से किसी का समय नहीं मापता।

भाग IX: संचालन

9.1. यदि डेटा किसी अलग रसीद अपडेट से पहले कमिट होता है, तो दोनों के बीच क्रैश होने पर पुरानी रसीद बची रहती है और रीट्राई प्रभाव को दोबारा लागू कर सकता है। यदि रसीद पहले कमिट होती है, तो डेटा राइट से पहले क्रैश होने पर रीट्राई उस डेटा को छोड़ सकता है जो कभी था ही नहीं। एक परमाणु (atomic) ट्रांज़ैक्शन दोनों अंतराल हटा देता है। ट्रांज़ैक्शन की पावती खो जाने पर फिर भी रखा गया रीट्राई चाहिए।

9.2. `ip_rules: {"10.20.0.0/16": "ALLOW"}` (या स्कोर 100 वाली `ip_reputation` पंक्ति) staging को प्रवेश देता है; नियम `{"name": "admin", "path": "/admin/*", "action": "CHALLENGE", "challenge": {"difficulty": 20, "algorithm": "posw"}}` कार्य माँगता है; `ALLOW` और `"limits": {"rate": 30, "window_seconds": 10}` के साथ `headers: {"X-Partner-Key": ".*"}` से मेल खाने वाला नियम पार्टनर को प्रति 10 सेकंड 30 अनुरोधों पर प्रवेश देता है। नियम का कोटा हर उस अनुरोध पर जाँचा जाता है जिससे वह मेल खाता है, प्रति-क्लाइंट डेमन सीमा (`--rate-limit`, `--rate-window`) के बाद, इसलिए उस सीमा को कम से कम उतना ही उदार होना चाहिए।

भाग X: संदर्भ

10.1. चैलेंज रिकॉर्ड में उस पते का कुंजी-युक्त फ़िंगरप्रिंट रहता है जिससे उसे मँगाया गया था; समाधान किसी नए पते से आता है, इसलिए `verify_and_mint` प्रमाण देखने से पहले ही `FingerprintMismatch` लौटा देता है। सबसे छोटा सुधार इंटरस्टिशियल में है: जिस `400` का शीर्षक `CLIENT FINGERPRINT MISMATCH` हो, उस पर त्रुटि दिखाने के बजाय नया चैलेंज लाएँ और फिर से हल करें, क्योंकि पहले किया गया कार्य स्थानांतरित नहीं किया जा सकता।

जाँच जारी रखें

किसी अस्थायी परीक्षण में एक समय पर एक ही अपरिवर्तनीय नियम बदलें। स्नैपशॉट की दोबारा जाँच हटा दें, घटना की रसीद छोड़ दें, या रेट अंतराल को नीचे की ओर पूर्णांकित करें। परीक्षण चलाने से पहले विफलता के क्रम का अनुमान लगाएँ। उपयोगी रिग्रेशन परीक्षण केवल सामान्य पाथ के आउटपुट की पुष्टि से अधिक करता है: वह दिखाता है कि छोड़ी गई शर्त क्यों आवश्यक है।

शब्दावली

प्रवेश (Admission)
किसी अनुरोध को ऑरिजिन तक फ़ॉरवर्ड करने का निर्णय। Sibuna में यह निर्णय वैध सत्र, `ALLOW` नियम, स्टैटिक बायपास पाथ या प्रतिष्ठा के `allow` प्रीफ़िक्स से होता है, और यह कभी WAF की अस्वीकृति को ओवरराइड नहीं करता।

Aho-Corasick ऑटोमेटन
एक परिमित ऑटोमेटन जो इनपुट पर एक ही पास में पैटर्न के किसी निश्चित समुच्चय में से कोई भी पैटर्न खोज लेता है। Sibuna failure link को एक सघन ट्रांज़िशन टेबल में समाहित कर देता है, ताकि हर बाइट की लागत एक टेबल लोड हो।

बैन टेबल (Ban table)
प्रतिबंधित क्लाइंट पतों के समाप्ति-समय सहित 4,096 lock-free स्लॉट, जिन्हें हनीपॉट और क्लस्टर में साझा प्रतिष्ठा लिखते हैं, और जिन्हें हर अनुरोध पर पढ़ा जाता है।

कैंपेन (Campaign)
सुरक्षा घटनाओं का एक समूह जिनकी पेलोड एम्बेडिंग एक-दूसरे से 0.35 की कोसाइन दूरी के भीतर हैं; यह एक ह्यूरिस्टिक समूहन है, साझा स्रोत का प्रमाण नहीं।

चैलेंज रिकॉर्ड (Challenge record)
70 अक्षरों का स्टेटलेस पहचानकर्ता जिसे क्लाइंट को हल करना होता है: संस्करण, एल्गोरिदम, कठिनाई, ओपनिंग, जारी करने का समय, क्लाइंट फ़िंगरप्रिंट, PRF नॉनस, नियम हैश और 16-बाइट का कुंजी-युक्त BLAKE3 टैग। इसे जारी करने से सर्वर पर कुछ भी नहीं लिखा जाता।

कठिनाई, वर्क बिट्स (Difficulty, work bits)
वह एकल पूर्णांक जो दोनों टियर का पैमाना तय करता है: Hashcash के लिए शुरुआती शून्य बिट की संख्या, या अनुक्रमिक कार्य के लिए ट्री की गहराई से तीन अधिक।

Edge
`--data-dir` (एक नोड पर एक SQLite फ़ाइल) के साथ, या `--database-url` और `--cluster-node` (क्लस्टर द्वारा साझा एक PostgreSQL डेटाबेस) के साथ Sibuna डिप्लॉयमेंट, जो दोनों में से किसी भी सतह को स्थायी नीतियाँ, साझा प्रतिष्ठा और घटनाओं की फ़ॉरेंसिक जाँच देता है।

इंजन स्लॉट (Engine slot)
वह स्थान जहाँ प्रकाशित नीति इंजन एक reference-counted पॉइंटर के रूप में रहता है। वर्गीकरण के दौरान वर्कर एक संदर्भ पकड़े रहते हैं; स्टोरेज थ्रेड अनुरोध पाथ से बाहर नया इंजन बनाता है और पॉइंटर बदलकर उसे प्रकाशित करता है, और पुराना इंजन तब मुक्त होता है जब उसका अंतिम पाठक उसे छोड़ देता है।

फ़िंगरप्रिंट (Fingerprint)
क्लाइंट पते और User-Agent का कुंजी-युक्त हैश, जिसे चैलेंज और टोकन में बाँधा जाता है ताकि दोनों में से किसी को किसी दूसरे क्लाइंट से रीप्ले न किया जा सके।

Forward auth
डिप्लॉयमेंट मोड जिसमें इनग्रेस Sibuna से पूछता है कि किसी अनुरोध को प्रवेश दे (`200`) या उसे चैलेंज करे, अस्वीकार करे या सीमित करे (`401`, `403`, `429`), और ऑरिजिन तक प्रॉक्सी स्वयं करता है।

Gate
वह सतह जिसमें proof-of-work प्रवेश, सत्र, नियम, प्रतिष्ठा, GCRA सीमाएँ और प्रतिबंध हैं, लेकिन एप्लिकेशन निरीक्षण नहीं (`--gate`)।

GCRA
Generic Cell Rate Algorithm: प्रति क्लाइंट एक सैद्धांतिक आगमन समय, जो N का बर्स्ट और फिर प्रति उत्सर्जन अंतराल एक अनुरोध स्वीकार करता है, बिना किसी विंडो सीमा के।

Hashcash
टियर एक: ऐसा नॉनस खोजें कि चैलेंज, एक कोलन और दशमलव नॉनस के SHA-256 में b शुरुआती शून्य बिट हों। अपेक्षित 2^b प्रयास, सत्यापन एक हैश से।

हनीपॉट (Honeypot)
`/__sibuna/honeypot` की ओर एक अदृश्य लिंक; जो क्लाइंट उसका अनुसरण करता है उस पर प्रतिबंध लगता है और प्रतिष्ठा -100 वाली एक घटना दर्ज होती है।

आइडल रीपर (Idle reaper)
एक थ्रेड जो `--idle-timeout` से अधिक समय तक चुप रहे कनेक्शनों को बंद करता है, ताकि कोई धीमा क्लाइंट किसी कनेक्शन थ्रेड को रोके न रख सके।

इंटरस्टिशियल (Interstitial)
संरक्षित पेज के स्थान पर दिया जाने वाला एम्बेडेड HTML पेज; यह एक चैलेंज लाता है, उसे Web Worker में हल करता है, समाधान पोस्ट करता है और पेज दोबारा लोड करता है।

JA4
FoxIO द्वारा परिभाषित 36 अक्षरों का एक फ़िंगरप्रिंट, जो बताता है कि कोई क्लाइंट TLS कनेक्शन कैसे खोलता है: प्रोटोकॉल, TLS संस्करण, सर्वर नाम, cipher और extension की संख्या, ALPN, और क्रमबद्ध cipher suites और extensions के हैश। Sibuna इसकी गणना नहीं करता; वह वह मान पढ़ता है जिसे कोई विश्वसनीय इनग्रेस `--ja4-header` द्वारा नामित हेडर में फ़ॉरवर्ड करता है, और नियम `ja4` फ़ील्ड से उसका मिलान करते हैं (SID 0014)। इसका चैलेंज और टोकन में बँधे क्लाइंट फ़िंगरप्रिंट से कोई संबंध नहीं है।

कुंजी-युक्त BLAKE3 (Keyed BLAKE3)
वह pseudorandom फ़ंक्शन जिस पर Sibuna का key schedule, चैलेंज टैग, सत्र टैग, फ़िंगरप्रिंट और चैलेंज नॉनस आधारित हैं।

MAC टोकन (MAC token)
डिफ़ॉल्ट सत्र कुकी: 40-बाइट पेलोड (संस्करण, वर्क लेवल, टाइमस्टैम्प, समाप्ति, नियम हैश, फ़िंगरप्रिंट) और 16-बाइट कुंजी-युक्त BLAKE3 टैग, जिसका सत्यापन स्थिर समय (constant time) में होता है।

वर्क लेवल (Work level)
वह तंत्र और वर्क बिट्स जिन्हें सत्र धारक ने वास्तव में हल किया, जो टोकन में रहते हैं; कोई रूट किसी सत्र को तभी प्रवेश देता है जब उसका लेवल रूट की आवश्यकता तक पहुँचता है।

ओपनिंग (Opening)
अनुक्रमिक कार्य के प्रमाण में, एक लीफ़ लेबल और रूट तक उसके पाथ पर पड़ने वाले सहोदर (sibling) लेबल; सत्यापनकर्ता पाथ की दोबारा गणना करता है और उसकी तुलना कमिटमेंट से करता है।

अनुक्रमिक कार्य का प्रमाण (Proof of Sequential Work, PoSW)
टियर दो: Cohen-Pietrzak हैश ग्राफ़, जिसके लेबल सभी पिछले लेबल पर निर्भर करते हैं, और जो $2^{n+1}-1$ अनुक्रमिक हैश अनिवार्य करता है, प्रूवर के पास चाहे जितने कोर हों। सत्यापन $t(n+1)$ हैश से।

प्रतिष्ठा trie (Reputation trie)
128-बिट radix trie जो नीति फ़ाइल से और `ip_reputation` टेबल से `allow`, `deny` और `challenge` प्रीफ़िक्स रखता है।

Robin Hood spent set
हल किए गए चैलेंज टैग की निश्चित क्षमता वाली open-addressed टेबल, जो किसी वैध समाधान को दो बार जमा होने से रोकती है।

नियम हैश (Rule hash)
: उस नियम के नाम का 64-बिट हैश जिसने चैलेंज माँगा, जो चैलेंज, टोकन और `X-Sibuna-Rule-Hash` हेडर में रहता है।

Shield
: डिफ़ॉल्ट सतह: Gate और साथ में सिमैंटिक फ़ायरवॉल (`--shield`)।

SID
: एक Sibuna Discussion रिकॉर्ड: `docs/sid/records` के अंतर्गत शोध-पत्र की शैली का डिज़ाइन दस्तावेज़, जिसमें सिस्टम के किसी एक भाग के पीछे की धारणाएँ, प्रमाण और मापन होते हैं।

सतह (Surface)
: उत्पाद के दो रूपों में से एक, Gate या Shield; स्टोरेज और क्लस्टरिंग दोनों के लिए विकल्प हैं।

WEIGH
: एक नियम एक्शन जो निर्णय लेने के बजाय अनुरोध के स्कोर में एक चिह्नित (signed) भार जोड़ता है; थ्रेशोल्ड के बराबर या उससे ऊपर के योग अतिरिक्त वर्क बिट्स के साथ चैलेंज करते हैं या अस्वीकार करते हैं।

Zaxonlite
: SID 0005 का एम्बेडेड SQLite रेप्लिकेशन इंजन (Multi-Paxos), जिसे SID 0011 में SQLite और PostgreSQL स्टोरेज से बदल दिया गया; उसके क्लस्टर विकल्प अब `RemovedOption` के साथ स्टार्टअप रोक देते हैं।

ग्रंथसूची

- Aho, A. V., and Corasick, M. J. “Efficient string matching: an aid to bibliographic search.” *Communications of the ACM* 18(6), 1975.
- ATM Forum. *Traffic Management Specification Version 4.0*, af-tm-0056.000, 1996 (Generic Cell Rate Algorithm)।
- Back, A. "Hashcash: a denial of service counter-measure." तकनीकी रिपोर्ट, 2002।
- Blocki, J., Lee, S., and Zhou, S. “On the security of proofs of sequential work in a post-quantum world.” *Information-Theoretic Cryptography (ITC)*, 2021.
- Celis, P. *Robin Hood Hashing*. पीएचडी शोध-प्रबंध, University of Waterloo, 1986।
- Cohen, B., and Pietrzak, K. “Simple proofs of sequential work.” *EUROCRYPT*, 2018.
- Dwork, C., and Naor, M. “Pricing via processing or combatting junk mail.” *CRYPTO*, 1992.
- Fiat, A., and Shamir, A. “How to prove yourself: practical solutions to identification and signature problems.” *CRYPTO*, 1986.
- Fielding, R., Nottingham, M., and Reschke, J. *HTTP Semantics*, RFC 9110, and *HTTP/1.1*, RFC 9112. IETF, 2022.
- FoxIO. *JA4+ Network Fingerprinting*. https://github.com/FoxIO-LLC/ja4, 2023 (JA4 TLS क्लाइंट फ़िंगरप्रिंट)।
- Hanson, N. “libinjection: SQL injection detection by tokenization.” Black Hat USA, 2012.
- Juels, A., and Brainard, J. “Client puzzles: a cryptographic countermeasure against connection depletion attacks.” *NDSS*, 1999.
- Lamport, L. "The part-time parliament." *ACM Transactions on Computer Systems* 16(2), 1998; और "Paxos made simple." *ACM SIGACT News* 32(4), 2001.
- Mahmoody, M., Moran, T., and Vadhan, S. “Publicly verifiable proofs of sequential work.” *Innovations in Theoretical Computer Science (ITCS)*, 2013.
- O'Connor, J., Aumasson, J.-P., Neves, S., and Wilcox-O'Hearn, Z. *BLAKE3: one function, fast everywhere*. विनिर्देश, 2020।
- Vyukov, D. "Bounded MPMC queue." 1024cores.net, 2011 (घटना कतार में उपयोग की गई sequence-stamped ring)।
- Wang, X., Hong, Y., Chang, H., Park, K., Langdale, G., Hu, J., and Zhu, H. “Hyperscan: a fast multi-pattern regex matcher for modern CPUs.” *NSDI*, 2019.
- Sibuna Discussions 0001 से 0014, `docs/sid/records`, 2026: प्रक्रिया, आधारभूत आर्किटेक्चर, घोषणात्मक (declarative) नीति इंजन, सिमैंटिक निरीक्षण, Zaxonlite स्टोरेज आर्किटेक्चर, गणितीय आधार, कंसोल प्रबंधन इंटरफ़ेस, AI बॉट ट्रैफ़िक निगरानी, chunked अनुरोध बॉडी, Rust पोर्ट, sqlx के माध्यम से स्टोरेज, rustls के माध्यम से कंसोल के लिए आउटबाउंड TLS, कंसोल स्टोरेज, और इनग्रेस से विश्वसनीय JA4 फ़िंगरप्रिंट।

अनुक्रमणिका

A
Aho-Corasick ऑटोमेटन, **135**

E
Edge, **135**

F
Forward auth, **135**

G
Gate, **135**
GCRA, **136**

H
Hashcash, **136**

J
JA4, **136**

M
MAC टोकन (MAC token), **136**

R
Robin Hood spent set, **136**

S
Shield, **137**
SID, **137**

W
WEIGH, **137**

Z
Zaxonlite, **137**

अ
अनुक्रमिक कार्य का प्रमाण (Proof of Sequential Work, PoSW), **136**

आ
आइडल रीपर (Idle reaper), **136**

इ
इंजन स्लॉट (Engine slot), **135**
इंटरस्टिशियल (Interstitial), **136**

ओ
ओपनिंग (Opening), **136**

क
कठिनाई, वर्क बिट्स (Difficulty, work bits), **135**
कुंजी-युक्त BLAKE3 (Keyed BLAKE3), **136**
कैंपेन (Campaign), **135**

च
चैलेंज रिकॉर्ड (Challenge record), **135**

न
नियम हैश (Rule hash), **137**

प
प्रतिष्ठा trie (Reputation trie), **136**
प्रवेश (Admission), **135**

फ
फ़िंगरप्रिंट (Fingerprint), **135**

ब
बैन टेबल (Ban table), **135**

व
वर्क लेवल (Work level), **136**

स
सतह (Surface), **137**

ह
हनीपॉट (Honeypot), **136**