SIBUNA

무할당, 작업 증명 기반 웹 방어 엔진

작업 증명 기반 요청 허용, 선형 시간 검사, 제한된 상태, SQL 기반 저장소

Vikrant Rathore

Ronak Rathore · Kanak Rathore 도움

Insan AI | 릴리스 1.0.0-beta.3

초록

웹 애플리케이션 방화벽과 엣지 방어는 대개 세 가지 아키텍처 문제를 겪습니다. 첫째는 비용 역전입니다. 송신자는 요청을 보내는 데 거의 아무 비용도 들이지 않지만, 방어자는 그 요청을 파싱하고, 백트래킹할 수 있는 정규 표현식으로 스캔하고, 외부 저장소에 조회합니다. 둘째는 실행 시간의 예측 불가능성입니다. 요청 경로의 힙 할당, 가비지 컬렉션 일시 정지, 여러 컨테이너로 이루어진 배포 때문에 지연 분포의 꼬리에 상한을 두기 어렵습니다. 셋째는 운영 부담입니다. 차단, 토큰, 제한을 위한 공유 상태는 흔히 운영자가 방화벽 옆에서 따로 운영해야 하는 서비스에 있습니다. **Sibuna**는 자체 완결형 바이너리 하나 안에서 몇 가지 단순한 메커니즘으로 이 문제들을 해결하려는 시도이며, 이 백서는 각 메커니즘의 근거를 설명합니다. (i) 작업 증명 기반 요청 허용: 브라우저는 규칙마다 정해지는 Hashcash 퍼즐이나 Cohen-Pietrzak 순차 작업 증명을 풀어 세션을 얻고, 서버는 저장한 적이 없는 챌린지에 대해 그 답을 70.2 ns 또는 22.3 us에 검증합니다. (ii) 상한이 있고 할당 없는 요청 경로: 바이트 클래스 Aho-Corasick 오토마톤은 봇 시그니처 40개를 55.7 ns에 대조하며, 이는 순차 검색보다 13.7배 빠릅니다. GCRA 속도 제한기는 클라이언트당 16바이트 상태로 6.2 ns에 판정하고, Robin Hood 사용 완료 집합은 풀이 하나를 21.0 ns에 기록합니다. SQL 인젝션, 크로스 사이트 스크립팅, 명령 인젝션, 경로 탐색 검사도 같은 패스에서 입력 길이에 선형인 시간에 실행됩니다. (iii) 자체 합의 프로토콜 없는 공유 상태: 노드 하나에는 내장 SQLite 파일을, 클러스터에는 공유 PostgreSQL 데이터베이스 하나를 쓰며, 요청이 결코 기다리지 않는 저장소 스레드 하나가 이를 읽습니다. AMD Ryzen 9 5950X 한 대에서 데몬을 CPU 네 개에 두었을 때, Sibuna는 초당 약 200,000개의 forward-auth 요청에 응답하고(워크로드에 따라 197,532에서 211,636) p99는 305에서 327 us입니다. 초당 100,000 요청의 개방 루프 부하에서는 p99.9를 62 us로 유지하며, 상주 메모리는 최대 약 16 MiB입니다. 각 수치를 어떻게 얻었는지 보고하고, 측정하지 않은 것도 밝힙니다.

6.2 ns	55.7 ns	22.3 µs	16 MiB
GCRA 속도 검사	**봇 시그니처 대조**	**PoSW 검증**	**데몬 최대 RSS**
해시된 클라이언트 | 샤드 16개	Aho-Corasick | 시그니처 40개	깊이 13 | 개방 16개	wrk 부하 | CPU 4개

AMD Ryzen 9 5950X 코어 하나(데몬은 CPU 네 개)에서 측정한 Sibuna, 2026년 10월 15일: benchmarks/results/primitives-latest.md와 product-latest.md. 측정 방법은 평가 절에 있습니다.

1 프롤로그: 요청의 경제학

서버에 도달하는 모든 요청은 서버의 주의를 요구합니다. 메서드, 경로, 헤더 몇 개만으로는 그 요청을 처리할 가치가 있는지 알 수 없습니다. 그래서 방어는 요청과 원본 서버 사이에 판단을 하나 둡니다. 허용하거나, 거부하거나, 먼저 검증 가능한 작업을 하라고 송신자에게 요구하는 것입니다. 이 백서는 Sibuna에서 이 판단을 수학에서 메모리의 바이트까지 따라갑니다.

> **Feynman의 물리적 직관: 작업 비용은 누가 치르는가?**
>
> 살짝 건드리기만 해도 도는 회전문을 생각해 봅시다. 그 뒤에는 경비원이 앉아 있는데, 회전문이 돌 때마다 일어나서 장부 세 권을 확인하고 본사에 전화해야 합니다. 밖에 있는 아이는 힘들이지 않고 오후 내내 회전문을 건드릴 수 있지만, 경비원은 점심 전에 녹초가 됩니다. 많은 방화벽이 이런 구조입니다. 비용이 적게 드는 쪽이 공격자입니다. 물리학은 더 나은 구조를 알려 줍니다. 회전문을 펌프에 연결해서, 지나가려는 사람은 누구든 물 한 양동이를 문 위의 물탱크로 끌어올리게 하는 것입니다. 끌어올리는 데는 작지만 정직한 노력이 듭니다. 경비원은 물이 올라왔는지 탱크를 흘끗 보기만 하면 됩니다. 흘끗 보는 데는 경비원에게 거의 비용이 들지 않지만, 끌어올리는 데는 방문자에게 실제 비용이 듭니다. 통근자는 한 번 끌어올리고 하루 종일 드나들며, 장난꾸러기는 곧 집에 돌아갑니다.

이 백서에서 인물의 이름을 단 상자는 Richard Feynman, Donald Knuth, Leslie Lamport의 설명 방식을 본받아 필자들이 직접 쓴 글입니다. 상자에 출처를 밝힌 경우가 아니면 인용문이 아닙니다.

자동화된 송신자는 요청 하나를 보내는 데 비용이 거의 들지 않습니다. 요청이 일반적인 방화벽에 도달하면 방어자는 TLS 핸드셰이크를 마치고, 요청을 복사하고 나누려고 메모리를 할당하고, 정규 표현식 목록을 실행하고, 카운터를 갱신하려고 공유 저장소까지 네트워크 왕복을 할 수도 있습니다. 이 비대칭에 보편적인 비율을 붙이지는 않습니다. 그런 비율은 없기 때문입니다. 비율은 원본 서버, 캐시, 공격에 따라 달라집니다. 대신 간단한 모델로 판단을 드러내 보겠습니다. c_o를 허용된 요청 하나에 대한 원본 서버의 작업, c_g를 요청 하나에 대한 게이트의 작업, a를 게이트가 허용하는 요청의 비율이라고 합시다. 게이트가 비용 이상의 가치를 하는 것은 다음 조건이 성립할 때뿐입니다.

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



그림 1 작업은 어디로 가는가. 위쪽의 먼저 검사하는 방어는 송신자가 보내는 모든 요청에 자기 작업을 씁니다. 아래쪽의 Sibuna는 송신자에게 기대 해시 시행 2^b번을 요구하고, 그 답을 해시 하나로 검증합니다. 작업량 막대는 모델이고, 검증 시간은 측정값입니다(16비트 Hashcash 검증, `benchmarks/results/primitives-latest.md`).

즉, 게이트가 원본 서버에서 덜어 주는 작업이 게이트 자신의 비용보다 클 때입니다[1]. 여기서 두 가지 결론이 나옵니다. 게이트 자신의 비용 c_g는 입력과 상관없이 작고 상한이 있어야 하며, 허용 결정은 서버가 아니라 송신자에게 비싸야 합니다.

Sibuna는 작업 증명을 요청 허용의 조건으로 삼아 이 비대칭을 뒤집습니다. 세션을 얻으려면 브라우저는 SHA-256 다이제스트의 선행 0비트가 b개인 nonce를 찾거나, 순차 작업 증명을 계산합니다. 기대 탐색량은 해시 시행 2^b번이며, $b = 16$이면 65,536번입니다. 서버는 후보 하나만 확인합니다. 서버의 확인에는 상태도 없습니다. 챌린지를 발급할 때는 아무것도 쓰지 않으며, 비용을 치른 풀이만 메모리를 차지합니다.

작업 증명은 사람임도, 신원도, 의도도 입증하지 않습니다. 송신자는 컴퓨터를 빌리거나, 세션 수명 동안 세션을 공유하거나, 작업을 여러 머신에 나눌 수 있습니다. 퍼즐은 요청 허용의 가격을 바꿀 뿐이며, 허용된 요청이 무엇을 할 수 있는지는 여전히 정책과 검사가 결정합니다.

2 설계 개요: 한 번의 패스에서 내리는 세 가지 판단

Sibuna는 요청 허용, 검사, 평판을 서로 다른 세 가지 판단으로 유지하고, 원본 서버가 어떤 작업도 하기 전에 한 프로세스 안에서 요청을 한 번 훑으며 이 판단을 내립니다. 이 기능은 두 가지 제품 운영 구성으로 제공합니다. *Gate*는 규칙, CIDR 평판, 로컬 폭주 제어를 갖춘 작업 증명 기반 요청 허용을 제공합니다. 기본값인 *Shield*는 여기에 상한이 있는 요청 페이로드 검사를 더합니다. 두 운영 구성 모두 원본 서버 앞의 리버스 프록시로 실행하거나, Caddy나 Nginx 같은 인그레스가 조회하는 forward-auth 엔드포인트로 실행할 수 있습니다. 정책, 평판, 인시던트 기록의 영속 저장은 어느 쪽에서든 선택할 수 있으며, 노드 하나에 로컬로 두거나 클러스터가 공유하게 할 수 있습니다[2].

2.1 설계 선택

아래 아이디어 대부분은 연구 문헌이나 이전 시스템에서 왔으며, 기술 안내서 제II부가 그 계보를 정리합니다. 이 작업이 시도한 것은 이 아이디어들을 상한이 있는 요청 경로 하나에 결합하고, 각 단계를 저렴하게 유지하고, 결과를 정직하게 측정하는 일입니다. 각 선택을 그 이유와 함께 나열합니다.

1. 발급에 서버 비용이 들지 않는 챌린지. 챌린지는 키 기반 해시로 봉인한, 스스로를 인증하는 레코드이므로 발급해도 상태를 쓰지 않습니다. 비용을 치른 풀이만 메모리를 차지하며, 그 풀이도 챌린지가 만료되면 사라집니다(알고리즘과 그 보장).

[1] 기술 안내서 제I부, "요청 허용은 경제적 결정입니다". 이 부등식은 측정이 아니라 용량 모델이며, 대역폭, 연결 상태, 클라이언트 지연을 고려하지 않습니다.

[2] SID 0002, "Decision"; 기술 안내서 제II부, "하나의 바이너리에 담은 세 가지 아이디어"; 기술 안내서 제VI부, "선언형 정책 엔진". 규칙별 예, 요구 티켓, 작업 수준 세션은 기술 안내서 제IV부에서, 적응형 기준값은 기술 안내서 제IV부와 SID 0006(개정 노트, Part H)에서 가져왔습니다.

표 1 두 가지 제품 운영 구성 (SID 0002, 기술 안내서 제II부)

기능	Gate (`--gate`)	Shield (`--shield`, 기본값)
요청 허용	Hashcash 또는 순차 작업 브라우저 챌린지, 클라이언트와 치른 작업에 묶인 세션 토큰	동일
정책	ALLOW, DENY, CHALLENGE, WEIGH 동작을 갖춘 선언형 규칙(경로, User-Agent, 헤더, IPv4와 IPv6 CIDR, 신뢰할 수 있는 인그레스가 보낸 JA4 TLS 지문), 봇 시그니처, 평판	동일
폭주 제어	GCRA 속도 제한, 허니팟, 임시 차단	동일
페이로드 검사	사용 안 함	본문의 처음 8 KiB에 대한 SQL 인젝션, 크로스 사이트 스크립팅, 명령 인젝션, 경로 탐색 검사
선택적 상태 저장	SQLite에 저장하거나 PostgreSQL로 공유하는 정책과 평판	여기에 인시던트 포렌식 추가: 전문 검색과 캠페인 군집화

2. 규칙마다 고르는 두 가지 작업 계층. Sibuna는 Hashcash와 함께 Cohen-Pietrzak 순차 작업 증명을 제공합니다. 이 증명은 양자 증명자에 대해서도 건전성이 증명되었고, 마이크로초 단위로 검증됩니다. 결제 경로는 순차 작업 20 작업 비트를, 블로그는 Hashcash 16비트를 요구할 수 있습니다(작업 증명: 병렬 탐색과 순차 작업).
3. 치른 작업을 기억하는 세션. 세션 토큰은 보유자가 푼 방식과 작업 비트를 기록합니다. 토큰은 그 이하를 요구하는 경로만 허용하므로, 값싼 세션으로 비싼 경로를 열 수 없습니다.
4. 대칭 토큰. 세션은 키 기반 BLAKE3로 봉인합니다. 검증에는 155.8 ns가 들어 Ed25519 서명 검증보다 작업이 약 260분의 1이며, 양자 탐색으로도 제곱 수준의 속도 향상밖에 얻지 못합니다.
5. 세션으로 건너뛸 수 없는 검사. 요청 허용과 검사는 같은 패스에서 실행되며, 유효한 세션은 챌린지만 충족합니다. 세션이 있어도 `q=' OR 1=1--`이 담긴 요청은 거부됩니다.
6. 구조를 인식하는 선형 시간 검사. 바이트 클래스 스캔 한 번, 오토마톤 패스 한 번, 작은 구조 토크나이저들이 백트래킹할 수 있는 정규 표현식 없이 입력 길이에 선형인 시간에 판정합니다.
7. 상한이 있는 상태와 키 기반 배치. 모든 테이블은 고정 용량과, 가득 찼을 때의 명시된 동작을 가집니다. 키는 키 기반 해시로 배치되므로 공격자가 특정 대상을 겨냥해 충돌을 일으킬 수 없습니다(키 기반 해싱과 보안 검토).
8. 요청당 할당 없음, 멈춤 없는 정책 갱신. 연결은 고정 버퍼 하나로 모든 판단을 처리하고, 새 정책 엔진은 어떤 요청이나 쓰기 쪽도 기다리게 하지 않고 게시됩니다(기계적 공감: 무할당과 상한이 있는 상태).
9. 부하를 따라가는 난이도. 초당 검증된 풀이 50개라는 기준선을 넘으면 비율이 두 배가 될 때마다 작업 비트가 하나씩, 최대 여섯 개까지 늘어납니다. 따라서 폭주 트래픽은 지수적으로 더 많은 비용을 치르고, 조용한 트래픽은 기본 비용을 유지합니다.
10. 인시던트에서 찾는 캠페인. 거부된 요청은 작은 임베딩과 함께 기록되고, 비슷한 인시던트는 캠페인으로 묶여 운영자가 콘솔에서 조사할 수 있습니다.

3 기계적 공감: 무할당과 상한이 있는 상태

> **Knuth의 기계적 정밀성: 캐시 라인과 구체 수학**
> 실시간 시스템의 안쪽 루프에서 범용 할당기를 호출하는 프로그래머는 설계의 일부를 낯선 사람에게 넘긴 셈입니다. 할당기는 평균적으로 빠를 수 있지만 그 비용은 힘의 이력에 달려 있고, 그 이력은 요청을 보내는 사람이 씁니다. 무엇이든 최적화하기 전에 먼저 세어야 합니다. 각 단계가 몇 바이트를, 몇 개의 캐시 라인을 건드리는지, 서로 의존하는 읽기가 몇 번인지 셉니다. 그다음에는 측정해야 합니다. 세는 것도 틀릴 수 있기 때문입니다. 필요한 모든 바이트가 첫 소켓을 열기 전에 정해진 자리를 갖고, 모든 루프가 그 수학이 요구하는 만큼만 정확히 일할 때 프로그램은 우아합니다.

3.1 무할당 불변식

요청 경로의 동적 할당은 하나하나가 서비스 거부 공격의 지렛대입니다. 며칠에 걸친 단편화, 할당기 락 경합, 할당을 유발하도록 만든 요청에 의한 증폭이 그렇습니다. Sibuna의 규칙은 분류, 검증, 프록시 인계를 힙 할당 없이 실행한다는 것입니다. 연결은 소켓이 살아 있는 동안 64 KiB 읽기 버퍼와 16 KiB 쓰기 버퍼를 소유하고, 연결이 시작될 때 한 번 할당하는 40 KiB 프록시 작업 공간을 가집니다. 나머지는 모두 이 버퍼를 가리키는 슬라이스입니다. 파서는 요청의 메서드, 경로, 쿼리, 헤더, 쿠키를 읽기 버퍼 안에서 찾아낼 뿐 하나도 복사하지 않습니다[3]. 구현의 타입 시스템은 파싱된 요청이 그 요청이 가리키는 바이트보다 오래 살 수 없게 강제하므로, 요청이 아직 버퍼를 읽는 동안에는 버퍼를 다시 채울 수 없습니다.

이 규칙은 요청 경로의 계약이지, 프로세스 전체에 대한 주장이 아닙니다. 스레드 생성, 시작 과정, 저장소는 할당을 합니다. 계약은 할당을 세는 테스트로 확인합니다. 데몬의 정상 상태 테스트는 할당 횟수를 세고, 프록시 테스트는 프록시를 거친 keep-alive 요청 500개 동안 할당이 하나도 없음을 확인합니다[4].

[3] 기술 안내서 제V부, "연결당 스레드 하나, 개수는 제한" 및 "연결당 버퍼 하나".
[4] 기술 안내서 제V부, "연결 루프"; `cmd/sibuna/tests/steady_state.rs`와 `crates/net/tests/steady_state.rs`.

3.2 상한이 있는 스레드와 재사용

수락된 연결은 각각 자기 스레드에서 처리되므로, 느린 클라이언트는 자기 자신만 늦추고 요청 경로는 동기식으로 유지됩니다. 이벤트 루프도 검토했지만 스레드 모델을 유지했습니다. 요청 경로의 속도는 제로 카피 파서, 연결별 고정 버퍼, 할당 없는 판단에서 나오며, 이벤트 루프는 이 중 어느 것도 개선하지 않습니다[5]. 연결 스레드 수는 `--max-connections`(기본값 1,024)로 제한되며, 이를 넘으면 accept 루프는 아무것도 생성하지 않고 503으로 응답합니다. 소켓 피어 주소 하나는 기본적으로 연결을 최대 64개까지 가질 수 있으며, IPv6 피어는 해당 /64 네트워크 단위로 셉니다.

스레드는 join할 수 있는 슬롯으로 이루어진 고정 테이블에서 나옵니다. 끝난 스레드는 join될 때까지 스택을 유지하므로, 새 작업은 커서를 앞으로 옮기는 대신 끝난 슬롯 가운데 가장 낮은 슬롯을 join해 차지합니다. 그러면 상주 메모리는 처리한 연결 수가 아니라 열려 있는 연결 수를 따라갑니다. 연결을 마친 스레드는 대기하다가 다음 연결을 맡습니다. 측정이 어떻게 이 마지막 규칙을 더하게 했는지는 측정이 바꾼 설계 절에서 설명합니다.

3.3 요청 처리 흐름

요청은 아래 검사를 정해진 순서로 거칩니다. 값싼 검사가 먼저 오며, 요청을 끝내는 판단은 그보다 비싼 단계가 실행되기 전에 요청을 끝냅니다.

유효한 세션은 챌린지를 충족할 뿐, 거부나 WAF 탐지 결과를 결코 덮어쓰지 않습니다. 한편 프록시는 원본 서버의 응답 헤드를 읽고, 프레이밍(본문 없음, 길이, 청크 인코딩, 연결 종료까지)을 분류하고, 본문을 정확히 중계한 뒤, 깨끗한 경계에서 끝난 원본 서버 소켓을 256개짜리 스트라이프 풀에 돌려줍니다. 그래서 프록시된 클라이언트 연결은 열린 채로 유지되고, 요청마다 원본 서버에 다시 연결하지 않습니다.

3.4 요청을 멈추지 않는 정책 게시

정책이 바뀌면 정책 엔진을 요청 경로 밖에서 다시 만들며, 요청이 절반만 만들어진 엔진을 보는 일은 결코 없어야 합니다. 저장소 스레드는 완전한 엔진을 만들어, 세대 카운터가 있는 슬롯에 참조 카운트와 함께 게시합니다. 각 연결은 캐시를 유지합니다. 세대를 acquire 읽기 한 번으로 읽고, 게시가 일어난 뒤에만 새 참조를 얻습니다. 옛 엔진은 그것을 붙잡고 있던 마지막 캐시가 놓을 때 해제되며, 쓰기 쪽이 읽기 쪽이 빠져나가기를 기다리는 일은 없습니다. 경합 상황에서 캐시된 읽기는 읽기-쓰기 락을 잡는 것보다 400배 넘게 저렴합니다. 엔진 자체는 약 200 KiB(209,052바이트)의 압축된 테이블을 담으며, 규칙 128개와 트라이 노드 8,192개를 수용합니다.

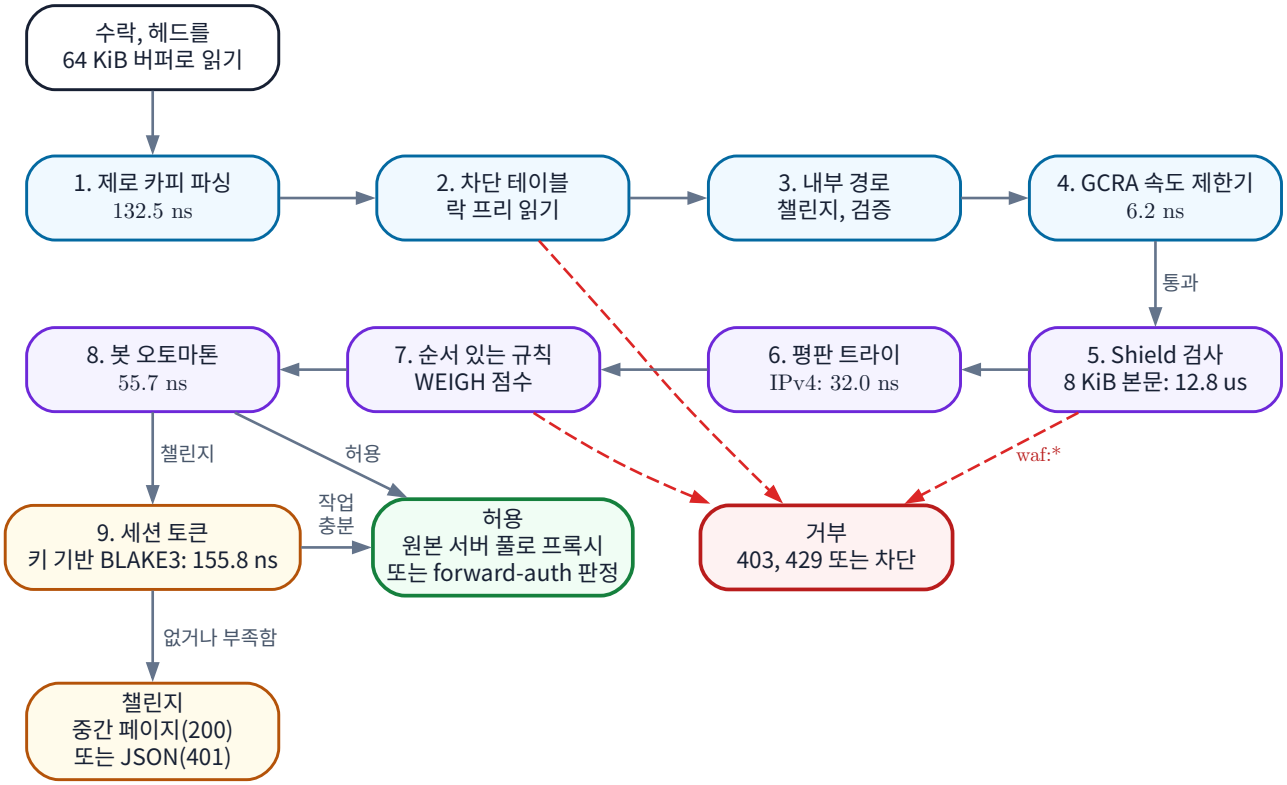


그림 2 요청 처리 흐름과, 기본 연산 측정 모음이 측정하는 기본 연산의 코어 하나 기준 중앙값 시간(`benchmarks/results/primitives-latest.md`)입니다. 단계는 번호 순서대로 실행되며, 점선 화살표는 조기 거부를 나타냅니다(GCRA 속도 제한기와 평판 트라이도 같은 방식으로 거부합니다). 브라우저 요청 하나의 전체 분류는 Gate 프로필에서 75.9 ns, Shield 프로필에서 790.8 ns가 걸립니다.

[5] SID 0010, "Concurrency model" 및 "Rationale and alternatives"; SID 0002, "Request contract" 및 개정 노트(주소당 연결 수, IPv6 접두사, 스트라이프 방식의 원본 서버 풀).

표 2 스레드 1개와 16개가 루프로 읽을 때 엔진 읽기 한 번의 비용, Ryzen 9 5950X (기술 안내서 제V부, 2026년 10월)

읽기 방식	스레드 1개	스레드 16개
읽기-쓰기 락, 읽기 가드만	4.2 ns	752 ns
읽기-쓰기 락과 새 참조	7.6 ns	833 ns
캐시된 참조와 세대 확인	0.8 ns	1.8 ns

표 3 샤드 락 처리량, 초당 백만 연산 단위, Ryzen 9 5950X (기술 안내서 제V부, SID 0010)

워크로드	적응형 뮤텍스	스핀 락
사용 완료 집합, 스레드 16개	10.8	8.6
사용 완료 집합, 스레드 64개	11.2	6.9
요청이 몰리는 주소 6개에 대한 GCRA, 스레드 16개	44.9	19.3
요청이 몰리는 주소 6개에 대한 GCRA, 스레드 64개	37.4	6.5

3.5 샤드별 락

연결 스레드가 공유하는 테이블(사용 완료 집합, 속도 제한기, 차단 테이블)은 키의 해시에 따라 샤드 16개로 나뉘며, 각 샤드는 자기만의 캐시 라인에 놓입니다. 임계 구역은 명령어 수십 개 길이이므로, 스핀 락과 적응형 뮤텍스(잠시 스핀한 뒤 잠드는 뮤텍스)를 비교 측정했습니다. 경합이 없을 때 스핀 락은 약 1나노초를 아끼지만, 엣지 서버는 CPU 수보다 많은 스레드를 실행하며, 보유자가 선점된 스핀 락은 모든 대기자가 자기 시간 조각을 태우게 만듭니다. 경합 상황에서 뮤텍스가 이겼으므로 뮤텍스를 유지했습니다.

4 알고리즘과 그 보장

요청 경로의 각 기본 연산에는 무엇을 대가로 무엇을 보장하는지에 대한 명제가 있습니다. 그중 세 가지를 정리로 제시하고 논증은 간략히 적습니다. 완전한 증명, 정의, 참고 문헌은 SID 0006과 Sibuna 기술 안내서 제III부부터 제VI부까지에 있습니다.

정리 1 (바이트 클래스 Aho-Corasick 오토마톤을 이용한 선형 시간 검사). $\mathcal{P}$를 전체 길이가 m이고 바이트가 $c \le 256$개 클래스에 속하는 패턴 집합이라 하고, 텍스트의 길이를 M이라 합시다. Sibuna는 오토마톤을 $O(mc)$ 시간과 공간에 만들고, 어떤 패턴이 텍스트에 나타나는지를 $O(M)$ 시간에 보고합니다. 입력 바이트마다 의존적인 테이블 읽기가 한 번이며, 이는 패턴 수와 무관합니다. 어떤 입력도 백트래킹을 일으킬 수 없습니다.

증명 개요. Aho와 Corasick의 오토마톤은 실패 링크가 있는 트라이를 만든 뒤 텍스트를 한 번 훑습니다. Sibuna는 빌드할 때 실패 링크를 전이 테이블에 접어 넣으므로, 입력 바이트 하나는 전이 하나이며 실패 사슬을 따라가는 일은 없습니다. 바이트 클래스는 어떤 패턴에 나오는 바이트 하나(한 글자의 대문자와 소문자는 같은 클래스) 또는 나머지 모든 바이트로 이루어진 단일 클래스입니다. 따라서 테이블 한 행은 256개가 아니라 c개 항목을 가집니다. 일치 상태는 다른 모든 상태 뒤에 번호가 매겨지므로, 일치 감지는 비교 한 번입니다.

측정. 봇 시그니처 40개에서 오토마톤은 55.7 ns가 걸려, 같은 패턴과 any-match 의미를 쓰는 순차 부분 문자열 검색의 763.0 ns보다 13.7배 빠릅니다. 바이트 클래스 덕분에 봇 시그니처 테이블 전체가 42,244바이트입니다[6]. □

정리 2 (GCRA 구간 상한). 창 W마다 요청 N개라는 제한을 정하고, 방출 간격을 $T = \max(1, \lceil W/N \rceil)$, 버스트 허용치를 $\tau = (N-1)T$라 합시다. 클라이언트는 이론적 도착 시각 TAT 하나를 유지합니다. 시각 t의 도착은 $\mathrm{TAT} \le t + \tau$일 때, 그리고 그때에만 적합하며, 그 뒤 $\mathrm{TAT} \leftarrow \max(\mathrm{TAT}, t) + T$로 갱신합니다. 그러면 적합한 도착으로 시작하는 길이 L의 모든 구간에는 적합한 도착이 많아야 $N + \lfloor L/T \rfloor$개 있습니다.

증명 개요. 적합한 도착에 시각 $t_1 \le \ldots \le t_k$의 순서대로 $1, \ldots, k$ 번호를 붙입니다. 적합한 도착은 매번 TAT를 적어도 T만큼 올리므로, j번째 도착 뒤에는 $\mathrm{TAT}_j \ge t_1 + jT$입니다. 도착 k가 적합했으므로 $\mathrm{TAT}_{k-1} \le t_k + \tau \le t_1 + L + \tau$입니다. 두 식을 합치면 $(k-1)T \le L + (N-1)T$, 즉 $k \le N + L/T$이고, k는 정수입니다.

구현. 클라이언트의 상태는 16바이트 셀 하나이며, 요청별 이력도 고정 창의 경계 효과도 없습니다. 샤드 16개는 키 기반 해시로 배치되는 셀 `--rate-table-cells`개(기본값 262,144개, 4 MiB)를 공유하며, 비워진 셀은 정리 스레드 없이 제자리에서 회수됩니다. 데몬이 이미 해시를 계산해 둔 클라이언트를 검사하는 데는 6.2 ns가 걸립니다[7]. □

[6] `benchmarks/results/primitives-latest.md` (55.7 ns와 763.0 ns); 테이블 크기는 기술 안내서 제VIII부 "자원 사용량"에 실린 `primitives-latest.json`의 값; SID 0006, Part F 및 바이트 클래스에 관한 개정 노트.

[7] SID 0006, Part D (Definition 22, Theorem 23); SID 0004, 개정 노트(키 기반 배치, `--rate-table-cells`); `benchmarks/results/primitives-latest.md` (`gcra_check_hashed`).

정리 3 (상태는 대가를 치른 작업으로만 늘어남). S를 데몬이 기억하는 챌린지 태그의 집합이라 합시다. 태그는 그 챌린지에 대한 유효한 증명이 받아들여진 뒤에만 S에 들어가고, 챌린지가 만료되면 빠져나갑니다. 챌린지 수명 하나 동안 받아들여진 서로 다른 풀이의 비율이 r이하이면 $|S| \le r \cdot \mathrm{TTL}$입니다. 가져갔지만 풀지 않은 챌린지는 메모리를 차지하지 않습니다.

증명 개요. 챌린지 식별자는 스스로를 인증하는 레코드입니다. 버전, 알고리즘, 난이도, 개방 수, 발급 시각, 클라이언트의 키 기반 지문, 의사 난수 함수로 도출한 nonce, 챌린지를 요구한 규칙의 해시로 이루어진 36바이트 뒤에 16바이트 키 기반 BLAKE3 태그가 붙으며, URL 안전 base64로 70문자입니다. 발급할 때는 아무것도 쓰지 않습니다. 검증은 태그를 상수 시간에 확인하고, 이어서 나이와 클라이언트 바인딩, 그다음 증명을 확인하며, 그 뒤에야 태그를 사용 완료 집합에 넣습니다. 키 모드 BLAKE3를 의사 난수 함수로 모델링하면, 검증 질의를 q번 하는 공격자가 태그를 위조할 확률은 많아야 $q \cdot 2^{-128}$에 PRF 이점을 더한 값입니다.

구현. 사용 완료 집합은 Robin Hood 개방 주소법을 쓰는 샤드 16개입니다. 만료된 항목은 자기 홈 슬롯에서 적어도 그만큼 멀리 떨어진 키로만 제자리에서 덮어쓸 수 있습니다. 그렇지 않으면 나중의 조회가 일찍 멈춰 살아 있는 태그를 놓칠 수 있습니다. 이 규칙 덕분에 정리 스레드가 필요 없고, 거부된 삽입이 살아 있는 태그를 내쫓는 일도 없습니다. 사용 처리와 조회를 합쳐 21.0 ns가 걸립니다[8].

□

4.1 의미 기반 방화벽

Shield는 경로, 쿼리, User-Agent, 구조적이지 않은 헤더, 본문의 처음 8 KiB를 검사합니다. 첫 버전은 교훈을 하나 남겼습니다. 그 버전은 평평한 부분 문자열 목록을 모든 헤더와 대조했는데, 목록에는 `/*`가 있었고, 모든 브라우저는 `Accept: text/html,...,*/*;q=0.8`을 보냅니다. 그래서 실제 브라우저가 모두 SQL 인젝션으로 분류되었습니다. 그 뒤로 탐지기는 구조에만 반응할 수 있으며, 완전한 Chrome 요청을 엔진에 통과시켜 깨끗하게 나와야 한다고 요구하는 테스트가 있습니다[9]. 검사는 네 단계로 이루어지며, 각 단계는 필드 길이에 선형입니다.

1. 바이트 클래스 사전 스캔. 256개 항목 테이블로 한 번 훑어 어떤 바이트 클래스가 나타나는지 기록합니다. 따옴표, `=`, `<`, 셸 구분자, `%`, NUL, 정규화 계기입니다. 이후의 모든 탐지기는 다시 스캔하지 않고 이 클래스를 참조하므로, 따옴표나 등호가 없는 필드는 SQL 토크나이저에 아예 들어가지 않습니다.
2. 태그가 붙은 시그니처 오토마톤. 모든 범주의 강한 시그니처가 오토마톤 하나에 들어 있습니다(정리 1). 일치하면 `waf:sqli`, `waf:xss`, `waf:path-traversal`, `waf:rce`라는 이름의 최종 거부가 됩니다.
3. 구조 토크나이저. SQL 토크나이저는 짧은 단어마다 키워드 테이블을 찾아보고, `or`와 `and` 뒤의 `literal = literal` 항진식을 확인합니다. 그 밖의 경우에는 따옴표가 반드시 있어야 하고 증거가 작은 점수에 도달해야 합니다. "it's a group order from the shop"이라는 문구는 3점을 받아 통과합니다. 크로스 사이트 스크립팅은 스크립트를 담을 수 있는 태그와 속성 위치의 이벤트 핸들러 속성을 함께 요구하고, 명령 인젝션은 구분자 뒤에 대입이 아닌 명령 이름이 오기를 요구합니다. 그래서 `a=1&id=5`는 명령이 아니라 매개변수입니다.
4. 정규화. 퍼센트 이스케이프, 주석 시작 기호, JSON이나 HTML 이스케이프, 접을 수 있는 공백이 있는 필드는 디코딩한 뒤 다시 검사합니다. 그래서 `1%27/**/UnIoN/**/SeLeCt`와 `%252e%252e/`는 원래 시그니처로 수렴합니다.

8 KiB 본문은 12,784.3 ns에 스캔되며, 바이트당 약 1.56 ns입니다. 운영자는 범주를 강제하는 대신 감사만 할 수 있으며, 그러면 탐지 결과를 기록하되 조치하지는 않습니다. 이때 필드는 범주별로 판정해야 합니다. 그래야 감사 대상 탐지가 강제 대상 탐지를 가리지 않습니다. Sibuna는 범주마다 필드를 한 번씩 스캔하는 대신, 시그니처가 나타나는 범주의 집합을 한 번의 스캔으로 구하고 그 가운데 원하는 범주만 찾습니다. SQL 인젝션을 감사 모드로 둔 같은 8 KiB 본문은 15,485.4 ns가 걸립니다[10]. Shield는 여전히 상한이 있는 휴리스틱 검사기이며 완전한 SQL 파서나 HTML 파서가 아닙니다. 검사 한도를 넘는 바이트는 검사하지 않습니다.

4.2 인그레스에서 받는 TLS 지문

User-Agent는 어떤 클라이언트든 마음대로 쓸 수 있는 문자열입니다. JA4 지문은 위조하기가 더 어렵습니다. JA4는 클라이언트가 TLS 연결을 여는 방식을 요약합니다. 프로토콜, 버전, 암호 스위트, 확장, 첫 번째 ALPN 값입니다. 클라이언트의 TLS 라이브러리는 첫 HTTP 바이트를 보내기 전에 이 값들을 정합니다. 핸드셰이크는 TLS를 종료하는 구성 요소만 보므로, JA4도 그 구성 요소만 계산할 수 있습니다. Sibuna는 인바운드 TLS를 종료하지 않으므로 JA4를 계산하지 않습니다. 대신 신뢰할 수 있는 인그레스가 계산해 `--ja4-header`로 지정한 요청 헤더에 담아 전달한 지문을 읽습니다[11].

- 신뢰. 이 헤더는 전달된 헤더를 신뢰할 때에만 읽으며, 신뢰 여부는 `X-Forwarded-For`에 적용되는 판단을 따릅니다. 신뢰할 수 없는 피어에서 온 헤더는 무시하고, 리버스 프록시 모드에서는 요청이 원본 서버에 도달하기 전에 이 헤더를 모두 제거합니다.
- 검증. 값은 36바이트 JA4 형식일 때, 그리고 그 이름의 필드가 정확히 하나 있을 때에만 받아들입니다. 형식이 잘못되었거나 반복된 값은 없는 것으로 취급하고 그 횟수를 셉니다.

[8] 기술 안내서 제III부(보조정리 3과 보조정리 4, 전송 형식); 기술 안내서 제V부, "Robin Hood 사용 완료 집합"; SID 0006, Part E (Lemma 28); `benchmarks/results/primitives-latest.md` (`robin_hood_spend_and_lookup`).

[9] SID 0004, "Revision note (2026-09-07)" 및 "Verification"; 기술 안내서 제VI부, "오탐의 교훈".

[10] `benchmarks/results/primitives-latest.md` (`8kb_body_semantic_scan`, `8kb_body_sqli_audit_other_enforce`); SID 0004, 개정 노트 "inspection modes"; 기술 안내서 제VIII부, "WAF 검사".

[11] SID 0014, "Specification" 및 "Security and operational considerations"; 기술 안내서 제IX부, "인그레스에서 받는 JA4 지문"; `.github/release-notes.txt`.

- 정책. 규칙은 지문 전체, 또는 *가 뒤따르는 접두사와 일치합니다. 지문이 없는 요청은 이런 규칙과 결코 일치하지 않습니다. 비교는 바이트 비교이며, 예열된 요청 경로는 여전히 힙 할당을 하지 않습니다.
- 기록. 인시던트는 해당 요청의 지문을 저장하고, 콘솔은 인시던트 증거에 지문을 보여 주며 지문으로 이벤트를 필터링합니다. 차단과 평판은 계속 주소를 키로 삼습니다. 정상 클라이언트 다수가 같은 TLS 라이브러리를 공유하기 때문입니다.

4.3 키 기반 해싱과 보안 검토

2026년 10월 요청 경로를 공격자의 관점에서 검토한 결과, 이미 해결되었다고 여긴 메커니즘에서 약점이 드러났습니다. 약점마다 수정하고, 그 내용을 해당 메커니즘의 설계 기록에 남겼습니다[12].

- 예측 가능한 테이블 배치. 속도 제한기와 차단 테이블은 소스 코드에 적힌 시드로 키를 해시해 배치했으므로, 누구나 피해자와 같은 탐사 윈도를 공유하는 주소를 계산할 수 있었습니다. 이제 배치에는 운영 체제의 난수에서 얻은 키로 계산하는 SipHash-1-3을 사용하며, 테이블마다 자체 키를 도출합니다. 데몬은 요청마다 클라이언트를 한 번만 해시하고(8.7 ns) 그 해시를 모든 테이블에 넘깁니다. 요청의 저장 경로는 43 ns에서 26 ns로 줄었습니다.
- 차단 축출. 유효한 차단은 결코 축출되지 않습니다. 키가 차지할 수 있는 모든 슬롯에 유효한 차단이 있으면 새 차단을 거부하고 그 횟수를 셉니다. 따라서 차단된 클라이언트가 자기 차단을 밀어낼 수 없습니다.
- 전달된 신원. `X-Forwarded-For`의 가장 왼쪽 항목은 클라이언트가 직접 쓸 수 있습니다. Sibuna는 오른쪽에서 `--trusted-hops`번째 항목, 즉 가장 가까운 신뢰할 수 있는 프록시가 덧붙인 항목을 사용합니다.
- 쿠키 토싱*(cookie tossing)*. 요청 하나에 세션 쿠키 값이 여러 개 담길 수 있습니다. 데몬은 최대 여덟 개까지 시도하므로, 형제 도메인이 심어 둔 쿠키가 방문자를 챌린지 루프에 가둘 수 없습니다.
- 사용 완료 집합의 용량. 태그 65,536개짜리 테이블은 5분인 챌린지 수명 동안 초당 약 220개의 검증된 풀이만으로 가득 찼고, 그 뒤로는 정직한 풀이가 거부되었습니다. 이제 용량은 설정할 수 있으며, 기본값은 태그 1,048,576개(24 MiB)입니다.
- 적응형 난이도. 부하가 높을 때 작업량을 올리는 제어기는 이제 발급된 챌린지 대신 검증된 풀이를 셉니다. 챌린지 발급은 요청을 쏟아붓는 쪽에 아무 비용도 들지 않기 때문입니다.

키 기반 배치에는 대가가 있으며, 이 대가를 테이블마다가 아니라 요청마다 한 번 치릅니다. 원시 클라이언트 바이트를 직접 해시하는 속도 제한 검사는 12.8 ns가 걸리고, 데몬이 쓰는 경로인 해시된 클라이언트에 대한 검사는 6.2 ns가 걸립니다. 속도 제한기와 사용 완료 집합은 0으로 채운 상태로 할당되므로 유휴 테이블은 메모리에 상주하지 않습니다. 따라서 설정한 용량은 실제로 쓰기 전까지 메모리를 차지하지 않습니다.

5 작업 증명: 병렬 탐색과 순차 작업

5.1 Hashcash 계층

첫 번째 계층은 비트 단위의 Hashcash입니다. 챌린지 문자열 C와 난이도 b가 주어지면, 클라이언트는 SHA-256 $(C \,\|\, \texttt{":"} \,\|\, \mathrm{dec}\,(N))$이 선행 0비트 b개를 갖도록 하는 nonce N을 찾습니다. nonce를 십진수로 표기하므로 WebAssembly 솔버, JavaScript 대체 솔버, 네이티브 검증기의 전송 형식이 똑같습니다. 이상적인 다이제스트라면 시행 횟수 K는 $p = 2^{-b}$인 기하 분포를 따릅니다.

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

탐색에는 두 가지 한계가 있습니다. 분산이 크므로 풀이 한 번이 느리다고 해서 워커가 고장 난 것은 아닙니다. 또 독립 시행은 병렬로 실행되므로, 기대 총작업량은 같더라도 코어가 많은 송신자가 더 빨리 끝냅니다[13].

5.2 순차 작업 계층

두 번째 계층은 Cohen과 Pietrzak(EUROCRYPT 2018)의 순차 작업 증명을 따릅니다. 라벨은 깊이 n인 완전 이진 트리를 이룹니다. 내부 노드는 두 자식을 해시합니다. 잎은 루트에서 내려오는 경로가 오른쪽으로 꺾인 곳마다 조상의 왼쪽 형제를 해시합니다. 이 왼쪽 형제 간선 때문에 계산은 순차적입니다. 잎은 그 왼쪽의 하위 트리가 모두 완성되어야 라벨을 붙일 수 있으므로, 루트 라벨은 $2^{n+1}-1$개의 라벨 전부에 순서대로 의존합니다. 루트 φ를 계산한 뒤 증명자는 Fiat-Shamir로 잎 t개를 도출하고, 잎마다 잎 라벨과 경로를 따라 있는 형제 라벨 n개를 보냅니다. 검증자는 개방된 잎을 각각 다시 계산하고 φ까지 해시해 올라갑니다. 비용은 고정된 작업 공간에서 라벨 계산 $O(tn)$번입니다.

매개변수는 수치가 작게 유지되도록 고릅니다. 작업 비트는 $n = b - 3$으로 깊이에 대응하며, $[4, 24]$ 범위로 제한됩니다. 증명의 크기는 $32(1 + t(n+1))$바이트이며, 깊이 13에 개방 16개라면 $32(1 + 16 \times 14) = 7200$바이트입니다. 증명자는 트리의 위쪽 몇 단계만 보관하고 개방할 하위 트리는 다시 계산하므로, 증명을 포함한 전체 작업 공간이 100 KB 미만입니다. 브라우저 탭과 네이티브 테스트는 같은 증명자를 사용합니다[14]. Blocki, Lee, Zhou(ITC 2021)는 양자 랜덤 오라클 모델에서 이 구성이 양자 증명자에 대해서도 건전함을 증명했습니다.

이 결과가 무엇을 입증하는지는 신중하게 보아야 합니다. 보안 결과는 각 논문의 랜덤 오라클 가정 아래에서, 논문이 제시한 구성과 상한에 대해 성립합니다. 우리의 구체적인 SHA-256 프로토콜, 매개변수 대응, 브라우저 풀이 시간을 독립

[12] SID 0002, 개정 노트(전달된 신원, 쿠키 값); SID 0004, 개정 노트(키 기반 배치, 차단 테이블, 측정한 비용, 0으로 채운 테이블); SID 0006, 개정 노트(사용 완료 집합의 용량, 적응형 제어기의 입력); SID 0010, 검토 이력(보안 검토, 2026-10-15); `benchmarks/results/primitives-latest.md` (`gcra_check`와 `gcra_check_hashed`).
[13] 기술 안내서 제III부, "Hashcash: 첫 번째 계층의 수학"; 기술 안내서 제I부, "확률변수로 보는 해시 탐색".
[14] 기술 안내서 제III부, "순차 작업 증명: 두 번째 계층", "증명자의 메모리", "매개변수는 시간이 아닙니다"; SID 0006, Part B 및 2026-09-07의 구현 검토.

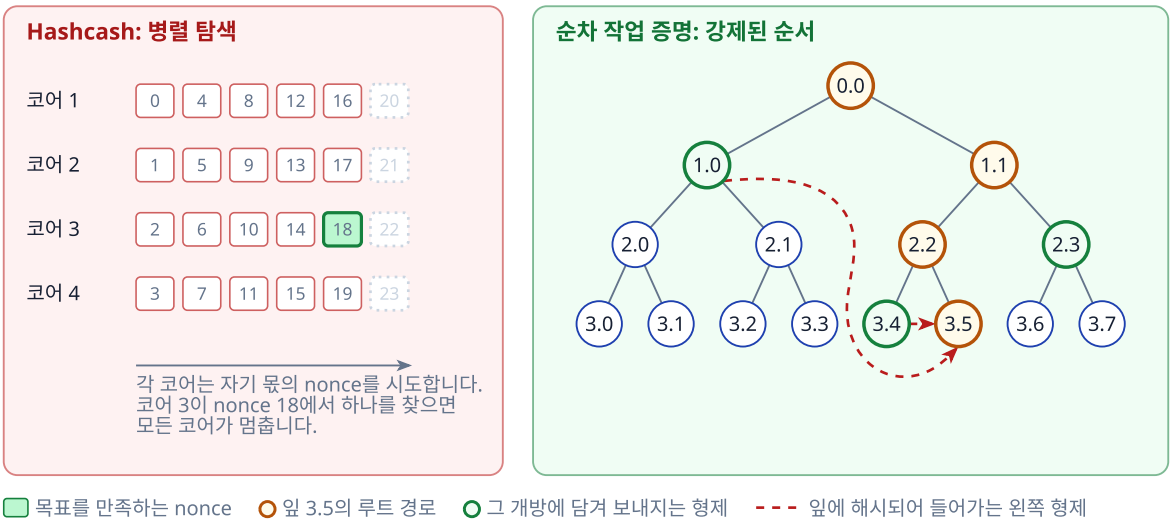


그림 3 병렬 탐색과 순차 작업. 왼쪽은 Hashcash입니다. nonce 시행이 서로 독립이므로 코어 M개는 약 M배 빨리 끝납니다. 오른쪽은 깊이 3인 Cohen-Pietrzak 트리입니다. 왼쪽 형제로 이어지는 점선 간선 때문에 잎 3.5는 왼쪽 하위 트리가 끝나기를 기다려야 하며, 잎 3.5의 개방은 그 라벨과 녹색 형제 세 개를 보냅니다(기술 안내서 제III부에서 가져옴).

적으로 감사한 결과는 아닙니다. 기본적인 표본 추출 상한 $(1-\alpha)^t$, 즉 서로 독립인 개방 t개가 고정된 불량 비율 α를 모두 피해 갈 확률은 개방이 많을수록 왜 도움이 되는지 설명합니다. 그러나 이 상한만으로는 루트를 많이 만들어 골라 보는 증명자에 대한 증명이 되지 않습니다. 부정행위가 결코 더 싸지 않다고 무조건 주장하지는 않습니다.

5.3 왕복 과정과 세션

유효한 세션이 없는 탐색 요청은 중간 페이지를 받습니다. 이 페이지에는 요구 티켓이 담깁니다. 이 결정이 요구하는 챌린지(알고리즘, 작업 비트, 개방 수)와 규칙의 해시를 이 클라이언트 전용으로 봉인한 것으로, 챌린지 수명 한 번 동안 유효합니다. 페이지는 이 티켓으로 챌린지를 가져오므로, 작업량은 그 작업을 요구한 결정이 정합니다. Web Worker가 WebAssembly 증명자로, WebAssembly를 쓸 수 없으면 JavaScript 대체 솔버로 챌린지를 풀고 답을 보냅니다. 성공하면 서버는 75문자의 세션 쿠키를 설정합니다. 이 쿠키는 키 기반 지문으로 클라이언트의 주소와 User-Agent에 묶이며, 페이지는 다시 불러옵니다. 브라우저의 증명자는 서버의 검증기와 같은 소스에서 컴파일되므로 둘이 서로 어긋날 수 없습니다[15].

5.4 검증 비용

두 계층 모두 SHA-256이 비용을 좌우합니다. Sibuna는 프로세서에 SHA 명령어가 있으면 실행 시점에 이를 선택하므로, 바이너리 하나가 이 명령어를 갖춘 모든 머신에서 빠르게 동작합니다. 16비트 Hashcash 검증은 70.2 ns, 깊이 13에 개방 16개인 증명의 검증은 22,272.4 ns가 걸립니다. 브라우저 증명자는 14,744바이트의 WebAssembly 모듈입니다. 종단 간 브라우저 검사에서 헤드리스 Chromium은 중간 페이지의 챌린지를 WebAssembly로 29에서 34 ms, JavaScript 대체 솔버로 84에서 91 ms 만에 풀었습니다[16]. 기본 연산 측정 모음은 검증만 측정합니다. 네이티브 측정 시간이 브라우저의 풀이 시간을 대신하지는 않습니다.

세션에는 대칭 토큰을 사용합니다. 검증자가 곧 발급자이거나 시드 하나를 공유하는 클러스터이므로, 키 기반 의사 난수 함수가 필요한 인증을 정확히 제공합니다. 키 기반 BLAKE3 토큰은 155.8 ns, Ed25519 토큰은 40,587.5 ns에 검증되므로, 서명 검증 한 번의 비용은 MAC 검증 약 260번과 비슷합니다. 이렇게 고른 이유는 하나 더 있습니다. Shor 알고리즘은 Ed25519의 바탕인 이산 로그 문제를 깨뜨리지만, 키 기반 해시에 대해 양자 탐색이 주는 이득은 많아야 제곱 수준의 속도 향상입니다. 토큰을 발급할 수 없는 검증자가 필요한 배포를 위해 Ed25519도 선택지로 남겨 둡니다.

[15] 기술 안내서 제IV부, "챌린지 왕복" 및 "세션 쿠키"; SID 0010, "The browser solver".

[16] `benchmarks/results/primitives-latest.md`; WebAssembly 크기는 기술 안내서 제VIII부 "자원 사용량"에 실린 `primitives-latest.json`의 값; 브라우저 풀이 시간은 SID 0010, 기준 3 (Chromium 155, `cargo xtask browser`); SHA 명령어의 실행 시점 감지는 SID 0006, 개정 노트(Part A부터 C까지).

6 자체 합의 프로토콜 없이 공유하는 상태

Lamport의 분산 불변식: 안전성, 활성, 그리고 로그의 주인

1987년 Leslie Lamport는 분산 시스템을 존재하는지조차 몰랐던 컴퓨터가 고장 나서 내 컴퓨터를 쓸 수 없게 되는 시스템이라고 표현했습니다. 이 말을 조언으로 읽으면, 우리가 의존하는 컴퓨터를 세어 보고 그중 어느 것이 고장 나더라도 성립해야 하는 것을 명시하라는 뜻입니다. 명제는 두 종류면 충분합니다. 안전성: 나쁜 일은 결코 일어나지 않습니다. 예를 들어 노드는 커밋된 트랜잭션이 쓴 적 없는 정책에 따라 동작하지 않습니다. 활성: 좋은 일은 결국 일어납니다. 예를 들어 데이터베이스에 닿을 수 있는 노드는 결국 최신 정책을 제공합니다. 합의 프로토콜은 이 둘을 함께 얻는 아름다운 방법입니다. 동시에 직접 떠안기에는 큰 시스템이기도 합니다. 그러므로 방화벽이 물어야 할 것은 합의가 좋은지가 아니라 로그를 누가 관리해야 하는지입니다.

6.1 왜 SQL인가, 왜 자체 합의 프로토콜이 아닌가

Sibuna가 저장하는 데이터는 많지 않습니다. 정책 행 수백 개, 평판 테이블, 그리고 운영자가 다른 시스템으로도 내보내는 포렌식 인시던트 정도입니다. 이런 상태에는 전용 복제 엔진이 필요하지 않습니다. 필요한 것은 믿을 수 있는 SQL 저장소, 즉 노드 하나라면 로컬 저장소이고 여러 노드라면 공유 저장소입니다. 그래서 단일 노드는 데이터 디렉터리의 내장 SQLite 파일에 WAL 모드로 상태를 보관하고, 클러스터는 모든 노드가 PostgreSQL 데이터베이스 하나를 가리키게 합니다[17].

우리는 방화벽 안에서 합의 프로토콜을 실행하지 않기로 했습니다. 합의 엔진은 자체 형식과 자체 장애 양상을 가진 큰 시스템이라 방화벽 옆에서 떠안기에는 부담이 크며, 종료 과정과 모든 업그레이드에 일을 더합니다. 그 대신 클러스터 상태의 가용성과 복제는 PostgreSQL이 맡고, 스트리밍 복제와 장애 조치 도구 또는 관리형 서비스가 이를 담당합니다. 우리는 이것이 더 겸손한 선택이라고 생각합니다. PostgreSQL의 복제는 우리 엔진이 거칠 수 있는 것보다 훨씬 많은 배포에서 검증되었고, Sibuna의 쓰기 빈도(일괄 기록하는 인시던트와 드문 정책 변경)는 주 서버 하나가 병목이 되지 않을 만큼 낮습니다.

저장소도 선택 사항입니다. `--data-dir`나 `--database-url` 없이 실행하면 데몬은 파일에서 정책을 읽는 인메모리 방화벽입니다.

6.2 저장소 스레드 하나

저장소 스레드 하나가 데이터베이스 연결을 소유하며, 그 밖에는 아무것도 SQL을 건드리지 않습니다. 요청 스레드는 결코 이 스레드를 기다리지 않습니다. 요청 스레드는 슬롯 512개짜리 크기가 제한된 링에 인시던트 레코드를 넣고(링이 가득 차면 레코드를 버리고 그 수를 셉니다), 게시된 엔진을 읽습니다. 저장소 스레드는 기본적으로 500 ms마다 틱을 한 번 실행합니다. 대기 중인 인시던트를 최대 32개까지 트랜잭션 하나로 쓰고, 정책 리비전을 읽어 바뀌었으면 엔진을 다시 빌드해 게시합니다. 재빌드는 로딩 전후에 리비전을 읽고, 그 사이에 다른 쓰기 주체가 커밋했으면 후보 엔진을 버리고 기존 엔진을 계속 씁니다. 운영자는 정책 테이블에 직접 쓸 수도 있습니다. 트리거가 쓰는 쪽의 트랜잭션 안에서 리비전을 올리고, PostgreSQL에서는 `NOTIFY`로 노드가 곧바로 틱을 실행할 수 있지만, 보장은 여전히 폴링이 맡습니다[18].

클러스터의 각 노드는 1부터 8,388,607 사이에서 서로 다른 `--cluster-node` 식별자를 가집니다. 챌린지 키는 발급한 노드에 묶이므로 풀이는 그 챌린지를 발급한 노드로 돌아가야 합니다. 반면 세션 토큰과 요구 티켓은 시드를 공유하는 모든 노드에서 유효합니다. 인시던트 식별자는 상위 비트에 노드를 담고, 각 노드는 일괄 쓰기와 같은 트랜잭션에서 갱신하는 영속 영수증을 유지합니다. 따라서 결과가 불분명한 장애 뒤에 같은 일괄 쓰기가 다시 실행되어도 아무것도 두 번 쓰이지 않습니다.

불변식: 저장소 계층 (SID 0011). **S1 (단일 소유자).** 모든 SQL은 저장소 스레드에서 실행되며, 어떤 요청 스레드도 데이터베이스를 기다리지 않습니다. **S2 (완전한 엔진).** 요청이 읽는 엔진은 모두 게시되기 전에 정책 테이블의 일관된 리비전 하나로부터 완전히 빌드된 것입니다. **L1 (최종적 게시).** 정책 테이블에 커밋된 변경은 데이터베이스에 닿을 수 있는 모든 노드에 다음 틱까지 도달합니다. **F1 (통제된 손실).** 데이터베이스가 내려가면 마지막으로 게시된 엔진이 계속 서비스하고, 인시던트는 크기가 제한된 링에 쌓였다가 링이 가득 차면 카운터를 올리며 버려지고, 저장소 스레드는 백오프하며 재시도합니다.

6.3 타입이 있는 스키마

스키마는 프로그램의 타입이 나타내는 것을 SQL로 나타내므로, 저장소 계층이 표현할 수 없는 행은 데이터베이스가 거부합니다. SQLite 테이블은 `STRICT`이고, 열거형 열에는 프로그램이 쓰는 이름을 정확히 나열한 `CHECK`가 있으며, 범위가 제한된 정수에는 범위 검사가 있고, 열은 값이 결코 빠지지 않을 때에만 `NOT NULL`입니다. 모든 문은 값을 매개변수로 바인딩하므로, 따옴표로 감싸 SQL을 조립하는 일이 없습니다. 콘솔의 저장소도 같은 규칙을 따릅니다. 타입이 있는 테이블 33개로 이루어진 기준 마이그레이션 하나가 있고 트리거는 없습니다. "토큰을 바꿀 때마다 감사 행을 쓴다" 같은 콘솔의 불변식은 각각 정확히 하나의 함수가 강제하며, 이 함수는 자기 문들을 트랜잭션 하나에서 실행합니다. SQLite와 PostgreSQL에서 똑같습니다[19].

[17] SID 0011, "Abstract", "Motivation and scope", "Why PostgreSQL for clusters"; 이전의 복제 설계는 SID 0005가 설명합니다.
[18] SID 0011, "The storage thread", "Policy revision", "Incidents and campaigns".
[19] SID 0011, "Schema and migrations" 및 검토 이력(전송 보안); SID 0013, "Abstract".

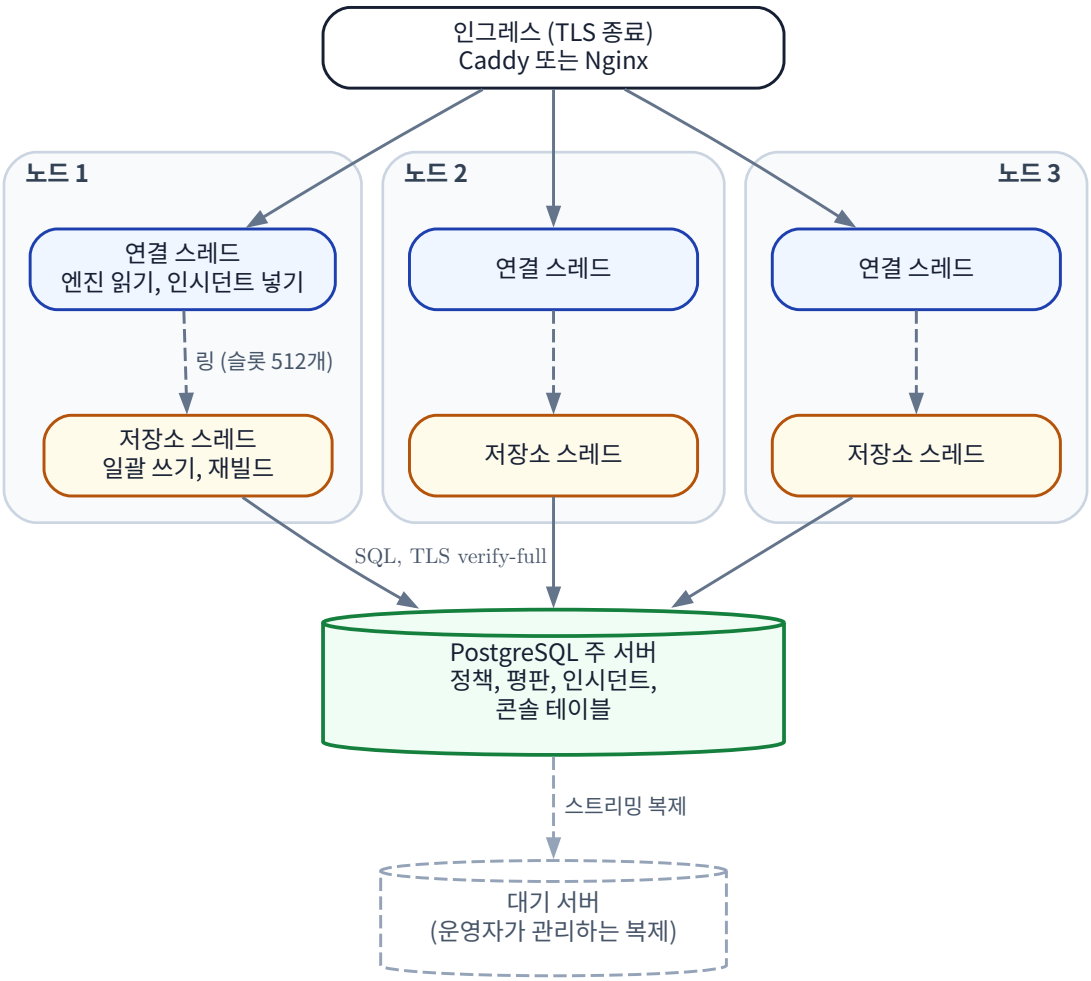


그림 4 Sibuna 클러스터. 모든 노드는 요청 경로를 메모리에 두고 SQL은 모두 저장소 스레드 하나에 맡깁니다. 노드들은 PostgreSQL 데이터베이스 하나를 공유하며, 그 복제와 장애 조치는 PostgreSQL이 담당합니다(SID 0011). 단일 노드는 그 대신 내장 SQLite 파일을 사용합니다.

데몬을 `postgres-tls` 기능으로 빌드하면 PostgreSQL 연결은 URL에 적힌 대로 libpq의 `sslmode`와 `sslrootcert`로 보호됩니다. 이 기능 없이 빌드한 데몬은 시작할 때 `require`, `verify-ca`, `verify-full`을 거부하며, 결코 평문으로 대체하지 않습니다. 운영 클러스터는 `sslmode=verify-full`을 사용합니다.

비슷해 보이는 인시던트는 캠페인으로 묶입니다. 각 인시던트에는 256바이트로 저장되는 64차원 임베딩이 있습니다. 저장소 스레드는 모든 노드의 최근 임베딩 16,384개를 메모리(4 MiB)에 두고, 가장 가까운 이웃과의 코사인 거리가 0.35 이하이면 인시던트를 그 이웃의 캠페인에 배정합니다. 더 오래된 인시던트와 비교해 군집화하지 않는 것은 의도적인 생략입니다. 캠페인은 본래 최근의 일이며, 이 상한 덕분에 메모리와 작업량이 고정됩니다.

6.4 이 설계가 하지 않는 것

이 선택의 한계는 분명히 밝혀 두어야 합니다. Sibuna에는 자체 합의 프로토콜이 없으며, 클러스터의 일관성은 PostgreSQL이 책임집니다. 데이터베이스를 잃은 노드는 다시 연결될 때까지 마지막 정책을 유지합니다. 한 노드에서 발급한 세션은 모든 노드가 받아들이고, 한 노드에서 사용된 풀이를 다른 노드에 재전송하면 거부됩니다. 노드들이 시드 하나를 공유하고 챌린지 키가 발급한 노드에 묶이기 때문이며, 두 성질 모두 데이터베이스가 필요하지 않습니다. SID 0011은 Committed 상태입니다. PostgreSQL 데이터베이스 하나를 쓰는 노드 세 개의 기능 테스트는 한 노드에 삽입한 정책을 세 노드 모두가 틱 한 번 안에 적용하고, 한 노드의 허니팟 차단이 다른 노드에 전달되며, 인시던트 식별자가 결코 충돌하지 않음을 보여 줍니다. PostgreSQL 데이터베이스 하나를 쓰는 여러 노드를 부하 상태에서 구동하는 하니스는 아직 없으므로, 이 논문은 클러스터 측정 결과를 보고하지 않습니다. 릴리스 실행 파일은 PostgreSQL 지원 없이 빌드됩니다. 클러스터를 구성하려면 Sibuna를 `postgres-tls` 기능으로 소스에서 빌드합니다[20].

[20]SID 0011, "Security and operational considerations" 및 "Acceptance criteria and implementation status" (기준 2, `crates/storage/tests/multi_node.rs`); `.github/release-notes.txt`, "Known limitations"; SID 0002, "Authentication and work" (노드에 묶인 챌린지 키); 기술 안내서 제VIII부, "측정하지 않은 것".

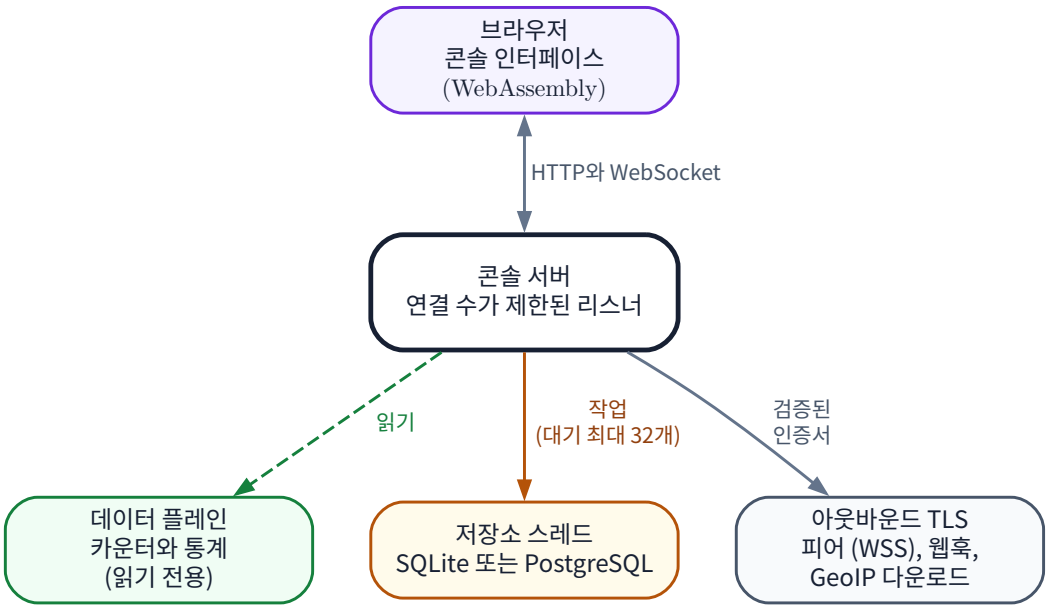


그림 5 콘솔. 브라우저는 인터페이스를 WebAssembly 모듈로 실행합니다. 콘솔 서버는 데이터 플레인의 카운터를 읽고 SQL은 저장소 스레드에 맡깁니다. 피어, 웹훅, GeoIP 배포처로 나가는 연결은 검증된 인증서와 함께 TLS를 사용합니다(SID 0007, 0012, 0013).

7 콘솔

Sibuna에는 명시적으로 켜야 하는 관리 콘솔이 들어 있습니다. 실시간 트래픽을 보여 주는 인증된 대시보드, 거부에서 기록된 증거까지 따라가는 조사 경로, 정책 편집, 주소 그룹, 공개 GeoIP 데이터를 이용한 국가 정보 보강, Argon2id 비밀번호와 TOTP 2단계 인증을 쓰는 사용자, 감사 기록, 웹훅, 클러스터 관리를 제공합니다. 콘솔은 같은 바이너리에 컴파일되어 들어가며, 그 인터페이스는 브라우저에서 WebAssembly 모듈로 실행됩니다[21].

콘솔을 규정하는 제약은 요청 경로와의 격리입니다. 콘솔은 데이터 플레인의 카운터와 데이터베이스를 읽고 데이터베이스에 쓸 수 있지만, 요청 스레드에 들어가지 않고, 요청 스레드에서 할당하지 않으며, 연결 스레드가 필요로 하는 락을 쥐지 않습니다. 리스너는 연결을 최대 96개까지 받으며, 그중 16개는 HTTP용으로 예약되고, 64개는 브라우저 스트림이, 16개는 피어 스트림이 쓸 수 있습니다. 1.0.0-beta.3에서 전체 인터페이스 모듈은 724,330바이트(gzip 압축 272,568바이트)입니다. 이는 빌드가 강제하는 상한 768 KiB보다 작고, SID 0007이 검토 대상으로 기록한 검토 기준 640 KiB보다는 큽니다[22].

콘솔은 오래 유지되는 피어 스트림, 웹훅, GeoIP 다운로드를 위해 아웃바운드 TLS 연결을 맺습니다. 인증서는 시스템 루트 인증서나 운영자의 CA 파일로 검증하고, 호스트 이름 검증은 끌 수 없으며, 평문 HTTP는 리터럴 루프백 주소에만 허용하고, 인증서 오류가 나도 결코 평문으로 대체하지 않습니다. Sibuna는 여전히 인바운드 TLS를 종료하지 않으며, 그 일은 계속 인그레스가 맡습니다.

SID 0007은 콘솔의 성능 목표를 정합니다. 콘솔을 켜서 생기는 비용은 데이터 플레인 처리량의 1% 이하, p99 지연의 10% 이하여야 합니다. 측정 결과는 하나 있습니다. 콘솔을 컴파일해 넣었지만 끈 경우, 저장소만 켠 경우, 켜 두었지만 유휴 상태인 경우 모두 전체 제품 워크로드 16개가 콘솔 없이 빌드한 데몬과 약 1% 이내의 차이에 머물렀으며, 이는 잡음 범위입니다. 부하 상태에서 활성 대시보드 8개로 진행하는 공식 관문은 아직 실행하지 않았으므로, 목표를 충족한다고 주장하지 않습니다. 실시간 대시보드 8개를 띄운 기능 실행은 통과했습니다.

8 실측 평가

8.1 방법

Sibuna는 `cargo xtask bench`로 실행하는 하니스 세 개로 측정합니다. 기본 연산 측정 모음은 시간을 재지 않는 예열 뒤에 시간을 재는 배치 일곱 개로 요청 경로의 연산을 측정하고, 변경 가능한 상태는 타이머 밖에서 다시 만들며, 연산마다 배치 중앙값과 함께 가장 빠른 배치와 가장 느린 배치를 보고합니다. 전체 제품 하니스는 forward-auth 모드와 리버스 프록시 모드에서, Gate와 Shield 운영 구성으로, 네 가지 워크로드에 대해 `wrk`(스레드 2개, 연결 64개, 5초, 반복 3회)로 데몬을 구동합니다. 워크로드는 `admitted`(유효한 세션), `challenged`(세션 없음), `allowed_static`(/robots.txt), `attack`(세션과 `q=' OR 1=1--`,

[21] SID 0007, "Abstract" 및 개정 노트(리스너의 요청 허용, 인터페이스 모듈, GeoIP와 비밀번호); SID 0010, 검토 이력(콘솔 포팅, TOTP); SID 0012; SID 0013. 격리 측정은 SID 0010의 검토 이력(콘솔 포팅)에 있으며, 기술 안내서 제VIII부 "콘솔 오버헤드"는 이 기록이 그 결과 파일을 명시하지 않는다고 적습니다.

[22] SID 0007, 검토 이력(Rust 포팅, 통합 정리, 데몬 통합). 이 기록은 SID 0014의 JA4 필드가 들어가기 전의 크기로 719,406바이트를 기록합니다. 1.0.0-beta.3의 크기는 데몬의 빌드 스크립트와 같은 방식(`cargo rustc --lib -p sibuna-console-ui --target wasm32-unknown-unknown --profile wasm -- -C link-arg=--compress-relocations`)으로 빌드한 모듈의 크기이며, 압축 크기는 `gzip -9`로 압축한 값입니다.

표 4 콘솔을 켠 SQLite 노드 하나에서 띄운 트래픽 대시보드 8개, 헤드리스 Chromium 155(benchmarks/results/console-8-dashboards-2026-10-15.md, 2026-10-15 기록). 결과: PASS

항목	값
데이터 플레인 요청	30.0초 동안 6,000개(초당 200개, 요청마다 연결 하나)
응답	6,000개: 200 x 1,200, 401 x 3,000, 403 x 600, 404 x 600, 500 x 600
오류(응답 없음)	0
지연 p50 / p90 / p99 / 최대	246 / 311 / 373 / 552 us (연결 시간 포함)
데몬 상주 메모리: 시작 시, 대시보드 8개 연결 시, 트래픽 후	36,152 KiB; 38,028 KiB; 41,496 KiB
대시보드	관리자 세션 3개와 벽면 디스플레이 5개. 각각 stats 델타를 134에서 139개 받았고 모두 같은 합계로 끝났으며, 예외, 콘솔 오류, Content-Security-Policy 위반은 없었음

Shield만 거부)입니다. 지연 하니스는 데몬이 어떻게 동작하든 고정 속도로 보내는 개방 루프 생성기를 사용하고, 각 요청의 시간을 보내기로 예정된 시각부터 재므로 데몬 안에서 대기한 시간은 데몬의 몫으로 계산됩니다[23].
실행은 2026년 10월 15일 호스트 한 대에서 기록했습니다. Linux 7.0.0-28을 실행하는 AMD Ryzen 9 5950X(코어 16개, 스레드 32개)입니다. 데몬은 워커 4개로 CPU 0부터 3에서, 부하 생성기와 스텁 원본 서버는 CPU 8부터 15(다른 CCD)에서, 기본 연산 측정 모음은 CPU 4에서 실행했습니다. 데몬은 SQLite와 콘솔을 컴파일해 넣은 기본 릴리스 빌드였으며, 저장소 없이 메모리에서 실행했습니다.

측정 범위. 모든 실행은 호스트 한 대에서 루프백으로 이루어집니다. TLS도, 광역 네트워크도, 실제 브라우저도 없습니다. 하니스는 할당 횟수를 세지 않으며, 할당 없는 요청 경로는 그 대신 정상 상태 테스트로 확인합니다. 하니스는 SHA-256으로 식별한 릴리스 바이너리를 수정 없이 실행합니다. 이 호스트에서 얻은 순위가 다른 호스트에서도 같은 순위를 뜻하지는 않습니다.

8.2 기본 연산 지연

표 5 Ryzen 9 5950X 코어 하나에서 측정한 Sibuna 기본 연산의 연산당 시간(나노초): 배치 일곱 개의 중앙값과 가장 빠른 배치, 가장 느린 배치(benchmarks/results/primitives-latest.md, 생성 2026-10-15T22:34:59Z)

하위 시스템	워크로드	중앙값	최소-최대
작업 증명	Hashcash 검증, 16비트	70.2	69.9-71.0
작업 증명	PoSW 검증, 깊이 13, 개방 16개	22,272.4	22,257.0-22,316.4
봇 매처	Aho-Corasick, 시그니처 40개	55.7	55.3-57.2
봇 매처	순차 부분 문자열 검색, 시그니처 40개(참고용)	763.0	762.2-763.5
IP 분류기	IPv4 CIDR 조회	32.0	31.8-32.2
IP 분류기	IPv6 CIDR 조회	79.3	78.5-79.6
토큰	키 기반 BLAKE3 세션 토큰	155.8	155.3-156.4
토큰	Ed25519 세션 토큰	40,587.5	40,544.6-40,631.1
사용 완료 집합	Robin Hood 사용 처리와 조회	21.0	20.8-21.3
속도 제한기	GCRA, 해시된 클라이언트	6.2	6.1-6.3
속도 제한기	GCRA, 규칙 범위 4개, 해시된 클라이언트	6.3	6.2-6.4
속도 제한기	GCRA, 바이트 형태의 클라이언트(키 기반 해시 포함)	12.8	12.7-12.8
속도 제한기	GCRA, 규칙 범위 4개, 바이트 형태의 클라이언트	13.1	13.0-13.2
HTTP 파서	제로 카피 요청과 쿠키	132.5	132.3-133.2
정책 엔진	전체 분류, Shield 프로파일	790.8	787.2-793.1
정책 엔진	분류, Gate 프로파일	75.9	75.6-76.1
WAF	8 KiB 본문, 의미 기반 스캔	12,784.3	12,771.5-12,809.2
WAF	8 KiB 본문, SQL 인젝션 감사 모드	15,485.4	15,460.9-15,530.1

[23] benchmarks/results/README.md, "Method" 및 "Provenance and limits" (rustc 1.99.0, 데몬은 cargo build --release -p sibuna, 49,113,856바이트); 기술 안내서 제VIII부, "방법론".
[24] 기술 안내서 제VIII부, "기본 연산 지연". primitives-latest.json에서 계산했습니다.

분포 폭은 좁습니다. 어느 행에서도 행 중앙값에서 4.2% 넘게 벗어난 배치는 없습니다[24]. 데몬은 요청마다 클라이언트를 한 번 해시하므로 해시된 클라이언트 행을 사용합니다. 원시 바이트를 넘기는 행은 검사마다 그 키 기반 해시를 포함합니다. 타이머는 개별 연산이 아니라 배치 전체를 감싸므로, 이 표는 연산별 백분위를 보고하지 않습니다.

8.3 전체 제품

표 6 `wrk` 부하를 받는 Sibuna: 초당 요청 수, 마이크로초 단위 지연, 마이크로초 단위 요청당 CPU 시간, 최대 상주 메모리. 실행 3회의 중앙값(`benchmarks/results/product-latest.md`, 생성 2026-10-15T23:15:24Z)

모드	운영 구성	워크로드(상태)	req/s	p50	p99	CPU	최대 KiB
forward-auth	Gate	`admitted` (200)	211,345	151	305	11.84	15,912
forward-auth	Gate	`challenged` (401)	198,315	160	326	13.15	16,176
forward-auth	Gate	`allowed_static` (200)	211,053	152	305	11.73	16,176
forward-auth	Gate	`attack` (200)	211,618	151	305	11.84	16,176
forward-auth	Shield	`admitted` (200)	208,101	152	308	12.87	15,928
forward-auth	Shield	`challenged` (401)	197,532	161	327	13.77	16,192
forward-auth	Shield	`allowed_static` (200)	211,636	151	305	11.78	16,192
forward-auth	Shield	`attack` (403)	210,549	152	307	12.32	16,192
리버스 프록시	Gate	`admitted` (200)	142,498	427	768	27.68	15,896
리버스 프록시	Gate	`challenged` (200)	199,465	160	324	12.98	16,160
리버스 프록시	Gate	`allowed_static` (200)	147,303	411	759	26.88	16,324
리버스 프록시	Gate	`attack` (200)	143,900	422	745	27.67	16,348
리버스 프록시	Shield	`admitted` (200)	138,965	438	857	28.57	16,064
리버스 프록시	Shield	`challenged` (200)	195,188	162	328	13.73	16,328
리버스 프록시	Shield	`allowed_static` (200)	145,673	415	746	27.27	16,328
리버스 프록시	Shield	`attack` (403)	209,507	151	306	12.17	16,328

forward-auth 모드에서는 인그레스가 Sibuna에 요청을 허용할지 묻고, 원본 서버는 없습니다. 모든 워크로드가 초당 197,532에서 211,636 요청 사이이며 p99는 305에서 327 us입니다. 이때 데몬의 CPU 네 개는 포화되지 않으므로(사용 코어 수 2.47에서 2.72), 한계를 정한 것은 부하 생성기로 보입니다. 리버스 프록시 모드에서 허용 행과 정적 행에는 스텁 원본 서버와 루프백 홉 하나가 더 포함되며, 데몬의 CPU가 포화됩니다(네 개 중 3.95에서 3.98 사용). Sibuna가 직접 응답하는 요청(챌린지, 그리고 Shield의 공격 거부)은 여전히 초당 약 195,000에서 210,000 요청으로 처리됩니다. 최대 상주 메모리는 모든 워크로드에서 15,896에서 16,348 KiB 사이입니다[25].

리버스 프록시 행 세 개에서 약 200만 응답 중 몇 개가 `503`이었습니다. 닫히는 연결이 주소당 연결 제한 안의 자리를 돌려주기 전에 `wrk`가 다시 연결했기 때문입니다. 이제 데몬은 연결을 닫는 응답을 쓰기 전에 자리를 돌려줍니다. 이 결과 파일들은 그 수정보다 먼저 기록되었습니다.

이후 릴리스 프로필을 바꾸어 번들된 SQLite의 디버그 정보를 제거했습니다. 빌드 매니페스트의 메모에 따르면 49 MB 가운데 약 13 MB입니다. 결과 파일은 그 뒤로 다시 생성하지 않았으므로 측정한 크기를 그대로 인용합니다[26].

표 7 Sibuna 데몬의 자원 사용량(`product-latest.json`과 `primitives-latest.json`, 2026년 10월 15일, 기술 안내서 제VIII부의 표)

항목	값
데몬 바이너리, SQLite와 콘솔을 포함한 기본 빌드	49,113,856바이트
유휴 상주 메모리, 전체 제품 실행	5,592~5,788 KiB
워크로드 중 최대 상주 메모리	15,896~16,348 KiB
워크로드 이후 상주 메모리	16,176~16,348 KiB
정책 엔진 테이블	209,052바이트
봇 시그니처 테이블	42,244바이트
WAF 시그니처 테이블	164,908바이트
브라우저 증명자(WebAssembly)	14,744바이트

[25] `benchmarks/results/product-latest.md`; 사용 코어 수는 `product-latest.json`의 `cores_busy_median`; `503` 응답과 그 수정은 `benchmarks/results/README.md`와 SID 0002, 개정 노트 (버그 수정).
[26] 기술 안내서 제VIII부, "자원 사용량"; 루트 `Cargo.toml`의 프로필 `[profile.release.package.libsqlite3-sys]`.

표 8 Sibuna Shield, 고정된 제공 요청률에서 허용된 요청: 지연(마이크로초)과 요청당 CPU 시간, 10초 실행 3회의 중앙값(benchmarks/results/latency-latest.md, 생성 2026-10-15T23:07:14Z)

모드	제공 req/s	달성 req/s	p50	p90	p99	p99.9	오류	CPU us
forward-auth	10,000	10,000	30	32	37	58	0	16.92
forward-auth	25,000	25,000	28	31	34	45	0	15.83
forward-auth	50,000	50,000	25	27	30	40	0	13.83
forward-auth	100,000	99,998	24	26	41	62	0	13.48
forward-auth	150,000	149,996	25	36	51	74	0	14.16
리버스 프록시	10,000	10,000	56	61	65	76	0	32.25
리버스 프록시	25,000	25,000	51	54	58	72	0	28.63
리버스 프록시	50,000	50,000	47	50	60	74	0	27.45
리버스 프록시	100,000	99,996	55	75	100	150	0	28.42
리버스 프록시	150,000	134,344	583,934	1,544,160	1,906,273	1,967,134	4	29.22

8.4 같은 제공 부하에서의 지연

wrk는 요청 64개를 항상 처리 중인 상태로 유지하므로 요청률이 데몬에 따라 달라지고, wrk의 백분위에는 응답을 기다리는 동안 보내지 못한 요청이 빠집니다. 개방 루프 하니스는 그 대신 제공 요청률을 고정하고 모든 지연을 데몬의 몫으로 계산합니다. 방문자가 실제로 겪는 것이 바로 이 지연입니다.

forward-auth 모드에서 p99.9는 초당 150,000 요청까지 75 us 미만입니다. 원본 서버가 경로에 있는 리버스 프록시 모드에서는 초당 100,000 요청에서 p99가 100 us, p99.9가 150 us입니다. 리버스 프록시 모드에서 초당 150,000 요청이면 데몬은 처리 용량을 넘어섭니다. 데몬은 초당 134,344 요청을 완료하는데, 이는 같은 워크로드에서 wrk가 측정한 138,965에 가깝습니다. 그리고 지연은 초 단위까지 늘어납니다. 용량을 넘어선 개방 루프 시험이라면 마땅히 보여야 할 모습입니다. 그 행의 오류 네 개는 과부하 중에 발생했습니다[27].

8.5 측정이 바꾼 설계

지연 하니스는 연결 스레드의 설계를 바꾸었습니다. 처음 실행에서는 데몬이 연결 64개를 각각 요청 4,096개 뒤에 닫았고, 초당 100,000 허용 요청에서 99.9번째 백분위가 0.3에서 2 ms였습니다. wrk에서는 보이지 않던 꼬리입니다. 스레드를 만들고 끝낼 때마다 시그널 스택을 매핑하고 해제하며 스택 페이지를 반환하는데, 이런 주소 공간 변경이 다른 스레드를 멈추게 했습니다. 실험 하나로 진단이 확정되었습니다. 연결을 전혀 닫지 않아 실행 중에 스레드를 하나도 만들지 않는 빌드는 같은 요청률에서 54에서 60 us를 보였습니다. 수정 후에는 연결을 끝낸 스레드가 대기 상태로 들어갔다가 다음 연결을 맡습니다. 10초 동안 대기한 스레드는 끝나므로, 버스트가 지나간 뒤에 스택이 메모리에 상주한 채 남지 않습니다. 수정 후 초당 100,000 요청에서 p99.9는 forward-auth 모드에서 62 us, 리버스 프록시 모드에서 150 us입니다[28].

드물게 주기적으로 일어나는 멈춤은 p99를 움직이기 훨씬 전에 먼 꼬리를 움직입니다. 연결 64개로 초당 100,000 요청을 보내면 연결 하나는 4096/1562.5 ≈ 2.6초 동안 유지되므로, 매초 스레드 약 24개가 시작되고 끝납니다. 요청 100만 개를 보내는 실행에서 상위 0.1%는 요청 1,000개뿐이므로, 스레드 교체마다 지연된 요청이 몇 개씩만 있어도 이를 채우기에 충분합니다. 이런 결함을 보려면 p99.9와 모든 요청에 대한 타이머가 필요합니다.

8.6 핵심 요약

1. 마이크로초 미만의 분류. Gate 프로필에서 정책 엔진은 브라우저 요청 하나를 75.9 ns에 분류합니다. Shield 프로필에서 전체 검사를 하면 790.8 ns가 걸립니다.
2. 대칭 검증. 키 기반 BLAKE3 세션 토큰은 155.8 ns에 검증되고, Ed25519 토큰은 40,587.5 ns가 걸립니다. 하나의 신뢰 도메인 안에서 같은 무결성을 약 260분의 1의 작업으로 얻습니다.
3. 작고 안정된 자원 사용량. CPU 네 개에서 초당 약 200,000 forward-auth 요청을 처리하며, 최대 상주 메모리는 모든 워크로드에서 16 MiB 근처이고 유휴 시에는 6 MiB 미만입니다.
4. 꼬리 지연에는 개방 루프 측정이 필요합니다. 스레드 교체로 인한 멈춤은 폐쇄 루프 하니스에서는 보이지 않았지만, 개방 루프 생성기 아래의 p99.9에서는 뚜렷하게 드러났습니다.

9 시장 비교: Sibuna와 이웃 도구

운영자가 Sibuna와 견주어 볼 가능성이 가장 큰 도구들과 Sibuna를 비교합니다. 다른 제품에 관한 사실은 2026년 9월에 공개 문서와 릴리스 산출물에서 확인했으며, Sibuna 기술 안내서 제II부에 출처와 함께 기록되어 있습니다[31]. 이 제품들은 저마다 사용자에게 충분한 가치를 제공하며, 우리 하니스가 측정하지 않는 장점을 지닌 설계를 선택했습니다. 여기서는 순위가 아니라 아키텍처의 차이를 설명합니다.

[27] benchmarks/results/latency-latest.md. 초당 25,000 요청의 행과 초당 150,000 요청의 리버스 프록시 행은 이 파일에 있습니다.
[28] SID 0002, 개정 노트(성능, 연결 스레드 재사용); 기술 안내서 제VIII부, "측정이 바꾼 설계"; benchmarks/results/latency-latest.md.
[31] 기술 안내서 제II부, "기능 비교" 및 "중간 페이지와 엣지"; Sibuna의 라이선스는 Cargo.toml에 있는 워크스페이스의 license입니다.

표 9 기능 비교(기술 안내서 제II부. 다른 제품에 관한 사실은 2026년 9월, JA4 행은 2026년 10월에 공개 문서에서 확인)

기능	Sibuna 1.0.0-beta.3	Anubis 1.27	SafeLine CE 9	Cloudflare WAF
실행 형태	바이너리 하나; 리버스 프록시 또는 forward-auth; 선택적으로 SQLite 파일 또는 공유 PostgreSQL 데이터베이스	Go 바이너리 하나; 리버스 프록시 또는 forward-auth	Docker 컨테이너 일곱 개(tengine, detector, mgt, luigi, fvm, chaos, PostgreSQL)	호스팅형 네트워크; DNS가 Cloudflare를 가리킴
런타임	네이티브 코드, 가비지 컬렉터 없음; 저장소를 끄면 타사 의존성 없음	Go, 가비지 컬렉션 사용	C/C++ 프록시와 탐지기, Go 관리 코드, PostgreSQL	독점 엣지 소프트웨어
라이선스	LGPL-3.0-or-later	MIT	GPL-3.0 관리 코드; 비공개 탐지기 이미지	독점 서비스
작업 증명 요청 허용	Hashcash(비트 단위)와 Cohen-Pietrzak 순차 작업, 규칙마다 선택	16진수 니블 단위의 SHA-256 Hashcash; 작업이 없는 챌린지	JavaScript 챌린지와 CAPTCHA; 작업 증명 방식 아님	관리형 챌린지, JS 챌린지, Turnstile
서버의 챌린지 상태	풀리기 전에는 없음; 풀린 태그는 고정 Robin Hood 테이블에 저장	저장소 백엔드: memory, bbolt, Valkey 또는 S3	스택이 관리	Cloudflare가 관리
세션 토큰	키 기반 BLAKE3, 75자, 치른 작업에 묶임; 선택적으로 Ed25519	Ed25519로 서명한 JSON Web Token; 선택적으로 HS512	스택이 발급하는 쿠키	`cf_clearance` 쿠키
페이로드 검사	Shield: 태그가 붙은 오토마톤과 구조 토크나이저, 본문의 처음 8 KiB	없음	OWASP 범주에 대한 의미 분석	관리형 규칙 집합(Pro 이상)
TLS 지문(JA4)	신뢰할 수 있는 인그레스가 전달하는 JA4 값을 규칙으로 비교; Sibuna는 계산하지 않음	문서화되지 않음; 정책에서 Anubis가 계산하는 HTTP 지문 JA4H를 참조할 수 있음	자체 TLS 종료 지점에서 계산; 사용자 지정 규칙과 내장 거부 목록	사용자 지정 규칙과 분석에서 JA3, JA4 사용(Enterprise, Bot Management 포함)
공유 상태	정책, 평판, 포렌식을 담는 PostgreSQL 데이터베이스 하나	공유 서명 키; 공유 Valkey 저장소	호스트당 스택 하나	전역 애니캐스트 네트워크
TLS 종료	아니요(인그레스)	아니요(인그레스)	예	예

9.1 Anubis와의 측정 비교

공식 Anubis v1.27.0 릴리스 바이너리를 전체 제품의 하니스로 Sibuna와 나란히 측정했습니다. 같은 워크로드 네 개, 모든 제품 스레드에 같은 CPU 네 개, 같은 `wrk` 부하, 맞춘 Hashcash 작업량(Sibuna는 0비트 8개, Anubis는 0인 16진수 자리 2개), 각 제품의 챌린지를 브라우저처럼 풀어 얻은 세션, 그리고 원본 서버로 `caddy respond`를 사용했습니다[29]. 여기서 Sibuna는 Shield 운영 구성으로 실행하므로 모든 페이로드도 검사하지만, Anubis는 페이로드를 검사하지 않습니다.

> **이 행을 읽는 법.** 이 수치는 모든 배포가 아니라 머신 한 대에서 실행한 워크로드 네 개를 설명하며, 몇몇 행은 같은 조건의 비교가 아닙니다. forward-auth 모드에서 Anubis는 `/robots.txt`와 공격 경로에 직접 응답합니다. Anubis는 페이로드를 검사하지 않으므로 Shield의 거부가 더 많은 일을 합니다. 챌린지 초기화는 Sibuna에서는 요청 두 개, Anubis에서는 페이지 하나입니다. 그리고 세션 확인 격차의 상당 부분은 대칭 토큰으로 설명됩니다. Anubis 자체의 HS512 방식도 기본값인 Ed25519보다 55% 빠르기 때문입니다. Anubis는 널리 쓰이고 기능이 충실한 도구이며, 관리자들은 타당한 이유로 다른 선택을 했습니다. 이 측정이 실제로 보여 주는 것은, 작고 상한이 있는 요청 경로가 방어자에게 요청 허용을 싸게 만들 수 있다는 점입니다.

forward-auth 모드에서 Anubis는 허용된 요청 하나에 CPU 106 us를 쓰고, Sibuna는 13 us를 씁니다. Anubis의 최대 상주 메모리는 forward-auth 모드에서 약 32 MiB였고 리버스 프록시 모드에서는 약 600 MiB까지 늘었지만, Sibuna는 내내 16 MiB 근처에 머물렀습니다. `caddy respond`를 원본 서버로 두면 Sibuna의 프록시 행은 부하 생성기와 CPU를 공유하는 원본 서버에 막힙니다. 전체 제품의 더 가벼운 스텁 원본 서버를 쓰면 같은 행이 초당 약 139,000 요청에 이릅니다.

9.2 아키텍처 상세 비교

9.2.1 SafeLine(Chaitin Technology)

SafeLine은 libinjection의 의미 기반 아이디어를 관리 콘솔을 갖춘 완전한 제품으로 발전시켰으며, 공급사는 탐지율 71.65%(균형 모드)와 76.17%(엄격 모드), 오탐률 0.07%와 0.22%를 보고합니다[32]. 이는 탐지 품질의 측정값이며, Sibuna는 자기 엔진에 대해 이런 측정값을 공개하지 않았습니다.

[29] `benchmarks/results/comparison-latest.md`와 `admission-latest.md`. 2026년 10월 16일 `cargo xtask bench product --anubis`와 `cargo xtask bench admission --anubis`로 기록했습니다. 기술 안내서 제VIII부 "Sibuna와 Anubis"에 방법과 두 제품의 모든 차이가 나와 있습니다.
[32] 기술 안내서 제VIII부, "측정하지 않은 것": 공급사가 보고한 수치이며, 2026년 9월에 확인했습니다.

표 10 wrk 부하에서 Sibuna(Shield)와 Anubis 1.27.0: 초당 요청 수와 p99 지연(마이크로초), 5초 실행 3회의 중앙값(benchmarks/results/comparison-latest.md). 따로 표시하지 않은 상태 코드는 200이며, Shield는 공격을 403으로 거부합니다.

모드	워크로드	Sibuna req/s	Sibuna p99	Anubis req/s	Anubis p99
forward-auth	admitted	208,635	308	36,509	7,477
forward-auth	challenged	193,087	342	113,170	3,781
forward-auth	allowed_static	212,444	304	89,904	4,547
forward-auth	attack	211,347 (403)	306	36,853	7,283
리버스 프록시	admitted	103,298	3,321	19,398	11,062
리버스 프록시	challenged	197,327	328	28,842	9,546
리버스 프록시	allowed_static	106,819	3,316	28,862	30,124
리버스 프록시	attack	210,483 (403)	306	23,399	14,124

표 11 CPU 두 개에서의 초당 요청 허용 연산 수, 배치 7개의 중앙값(benchmarks/results/admission-latest.md). Anubis는 토큰 방식마다 한 번씩 실행합니다.

제품(토큰)	세션 확인	미인증 확인	챌린지 초기화	증명 검증	상주 메모리
Sibuna(키 기반 BLAKE3)	88,104	78,997	42,664	57,924	7.6 MiB
Anubis(Ed25519)	10,951	28,619	7,893	12,668	32.6 MiB
Anubis(HS512)	16,976	27,785	7,952	19,261	29.1 MiB

- 배포. 커뮤니티 에디션은 CPU 코어 1개, 메모리 1 GB, 디스크 5 GB 이상인 Linux 호스트에서 Docker 컨테이너 일곱 개로 제공됩니다. TLS를 직접 종료하며, Sibuna는 그렇게 하지 않습니다.
- 상태. 공격 로그를 스택 안의 PostgreSQL에 보관하며, 호스트당 스택 하나입니다.
- *Sibuna*와 다른 점. Sibuna는 선택적인 SQLite 파일로 시작하는 바이너리 하나이며, 클러스터는 운영자가 이미 운영하고 있을 수 있는 PostgreSQL 데이터베이스 하나를 공유합니다. 전체 제품 실행에서 유휴 상주 메모리는 5,592에서 5,788 KiB였습니다. 우리 호스트에는 Docker 데몬이 없어 SafeLine을 실행하지 않았으므로, 성능은 비교하지 않습니다.

9.2.2 Anubis와 Coraza

작업 증명 중간 페이지는 2025년에 흔해졌습니다. 언어 모델 학습용 스크레이퍼 트래픽 때문에 평범한 원본 서버도 운영 비용이 커졌기 때문이며, Anubis는 그중 가장 잘 알려진 오픈 소스 구현입니다. Anubis는 기능이 충실하고 널리 배포된 Go 리버스 프록시입니다. JavaScript 퍼즐을 제공하고, JSON Web Token에 Ed25519로 서명하며, 챌린지 상태를 저장소에 보관합니다. Coraza는 ModSecurity 규칙 언어(SecLang)를 구현하고 OWASP Core Rule Set을 실행합니다. 이런 게이트 옆에 규칙 기반 검사를 더하는 한 가지 방법입니다.

- 작업은 어디로 가는가. Anubis와 Sibuna Gate는 모두 원본 서버가 무슨 일이든 하기 전에 클라이언트가 먼저 비용을 치르게 합니다. Sibuna는 더 강한 게이트가 필요한 경로를 위해 규칙마다 선택하는 두 번째 계층인 순차 작업을 더하며, 세션은 얼마만큼의 작업을 치렀는지 기억합니다.
- 서버가 기억하는 것. Anubis는 챌린지를 저장합니다. Sibuna는 스스로 인증되는 레코드를 발급하고, 풀린 태그만 고정 크기 테이블에 기억합니다(정리 3). 기본 토큰은 대칭 방식이라 검증 비용이 더 낮고, 양자 공격자에게도 제곱 수준의 속도 향상만 허용합니다.
- 검사. Sibuna Shield는 요청 허용과 같은 패스에서 자체 구조 검사를 수행하므로, 프로세스 하나와 파서 하나가 두 역할을 함께 맡습니다. OWASP Core Rule Set은 실행하지 않습니다. libmodsecurity와 Coraza 바인딩을 포함해 기존 SecLang 엔진을 검토했지만 어느 것도 채택하지 않았습니다. 엣지 서버의 검사 엔진은 보안에 결정적인 코드이므로 직접 소유하고 검토하는 편을 택했습니다. Sibuna가 Core Rule Set 호환성을 더한다면, 먼저 설계 기록에서 지원할 부분 집합을 명시할 것입니다[33].
- 측정. 같은 부하에서 호스트 한 대로 측정했을 때 Sibuna는 forward-auth 모드에서 허용된 요청을 다섯 배 넘게 처리했으며, p99는 더 낮았고 요청당 CPU는 약 8분의 1이었습니다(Anubis와의 측정 비교).

9.2.3 클라우드 엣지 WAF(Cloudflare 등)

호스팅형 엣지는 전역 제어 평면을 두고 여러 거점에 검사, 속도 제한, 봇 점수를 배치합니다. 서버 한 대로는 막을 수 없는 대규모 트래픽 공격을 흡수하며, 운영자 쪽 하드웨어도 필요하지 않습니다. Sibuna의 클러스터 배포는 그 형태, 즉 로컬 불변 스냅숏을 공급하는 공유 정책 평면을 빌리면서도 셀프 호스팅을 유지합니다.

- 데이터 레지던시. 호스팅형 WAF는 운영자의 TLS를 종료하고 공유 인프라에서 요청 페이로드를 보므로, 데이터 레지던시 요구 사항과 충돌할 수 있습니다.
- 요금제와 할당량. 기능은 요금제에 묶여 있습니다. Cloudflare가 공개한 할당량에 따르면 Free, Pro, Business, Enterprise 요금제에서 사용자 지정 규칙은 각각 5, 20, 100, 1,000개, 속도 제한 규칙은 각각 1, 2, 5, 100개입니다.

[33] SID 0010, “The OWASP Core Rule Set”.

- *Sibuna*와 다른 점. Sibuna는 요청당 요금 없이 운영자 자신의 머신에서 실행되며, 정책 변경은 다음 저장소 틱(기본 500 ms)에 모든 노드에 도달합니다. 대규모 네트워크 트래픽 공격을 완화하지 않고, 학습된 모델로 봇 점수를 매기지 않으며, TLS를 종료하지도 않습니다. 이런 기능이 필요한 배포라면 여전히 Sibuna 앞에 인그레스나 엣지 네트워크를 두어야 합니다.

10 구현: 작은 크레이트와 적은 의존성

Sibuna는 Rust로 작성되었으며, 의존성이 한 방향으로만 향하는 작은 크레이트들의 워크스페이스로 이루어집니다. 재사용 가능한 라이브러리, 그다음 설정과 진단을 담은 코어, 그다음 애플리케이션 라이브러리, 마지막으로 데몬과 브라우저 증명자 순입니다. 운영자에게 중요한 엔지니어링 선택 세 가지를 간단히 기록합니다[34].

적은 의존성. 우리는 모든 타사 라이브러리를 부채로 봅니다. 검토하고, 갱신하고, 신뢰해야 하기 때문입니다. 데이터 평면은 타사 라이브러리를 하나도 쓰지 않습니다. 해싱, 서명, base64, JSON, DEFLATE, Aho-Corasick, HTTP/1.1 파서는 직접 만든 작은 라이브러리이며, 다른 프로젝트도 따로 가져다 쓸 수 있습니다. 의존성은 성숙하고, 자체 완결적이며, 정말로 큰 문제를 해결할 때만 받아들이고, 그 이유를 빌드 매니페스트와 설계 기록에 함께 적습니다. 지금까지 받아들인 의존성은 두 개이며, 각각 라이브러리 하나 안에만 머뭅니다. SQLite와 PostgreSQL 저장소를 위한 `sqlx`는 전이 크레이트 약 100개를 가져오며[35], 콘솔의 아웃바운드 TLS와 PostgreSQL의 TLS를 위한 `rustls`는 실행 시점에 크레이트 13개를 씁니다. TLS 클라이언트를 직접 작성하면 Sibuna의 데이터 평면보다 큰, 보안에 결정적인 프로젝트가 되기 때문입니다[36]. 저장소와 콘솔이 없는 빌드에는 타사 의존성이 전혀 없습니다.

메모리 안전성. 컴파일러의 메모리 안전성 검사를 우회하는 코드는 기본적으로 금지되며, 그런 코드가 필요한 모듈은 이름을 밝혀 명시적으로 허용받고 사용처마다 주석으로 근거를 적어야 합니다. 그런 모듈은 몇 개뿐입니다. 프로세서의 SHA 명령어를 쓰는 경로, 비밀을 지우는 쓰기 하나, 시그널 처리를 위한 C 라이브러리 함수 몇 개의 선언, 두 WebAssembly 모듈의 export, 그리고 저장소 스레드로 인시던트를 나르는 링입니다. 링은 속도 때문에 그런 코드를 남겨 둔 유일한 곳이며, 그것도 측정을 거친 뒤였습니다. 생산자가 16개일 때 표준 라이브러리의 크기 제한 채널은 push 한 번에 20에서 45% 더 많은 비용이 들었습니다[37]. 그 밖의 곳에서는 직접 작성한 메모리 관리가 했을 일을 소유권이 대신합니다. 파싱된 요청은 버퍼보다 오래 살 수 없고, 엔진은 마지막으로 읽는 쪽과 함께 해제되며, 소켓은 어떤 스레드든 아직 닿을 수 있는 동안에는 닫힐 수 없습니다.

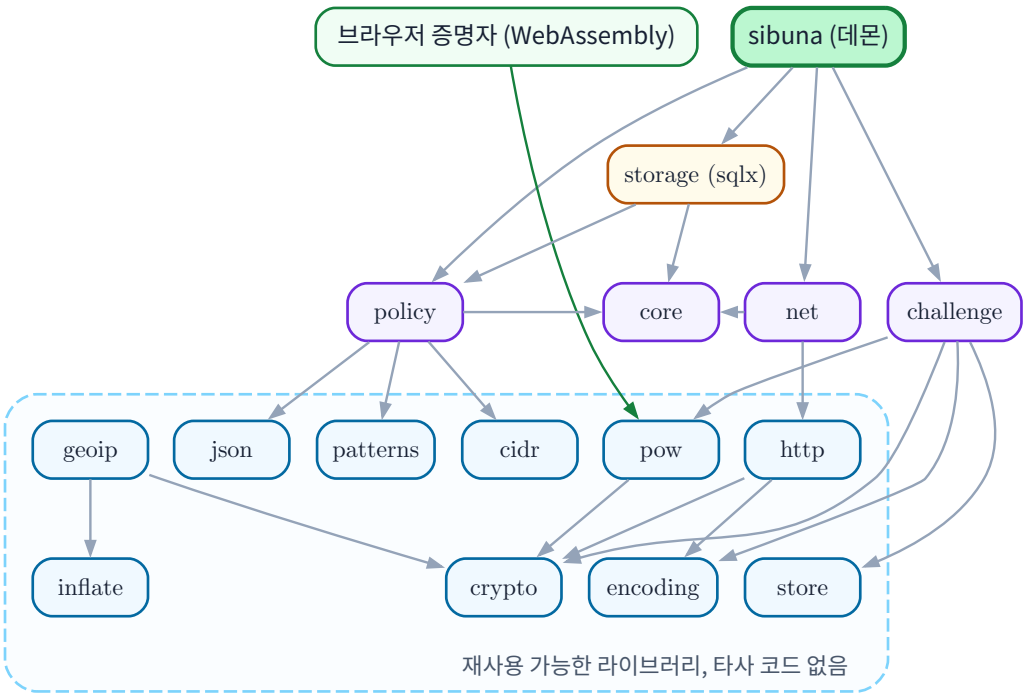


그림 6 Sibuna의 라이브러리 구조를 단순화한 그림(SID 0010을 따름). 화살표는 라이브러리에서 그 라이브러리가 사용하는 쪽으로 향합니다. 재사용 가능한 라이브러리는 어떤 애플리케이션 라이브러리에도 의존하지 않으며, 타사 코드를 가져오는 것은 저장소 라이브러리와 콘솔 라이브러리뿐입니다.

[34] SID 0010, "Workspace", "Dependency policy", "Engineering standards".
[35] SID 0011, "Why sqlx"; 타사 크레이트마다 채택 이유를 적어 둔 `Cargo.toml`.
[36] SID 0012, "Specification", "Rationale and alternatives" 및 검토 이력(PostgreSQL TLS 구현).
[37] SID 0010, 검토 이력(보안 검토, 2026-10-15): 표준 크기 제한 채널과 Vyukov 링의 비교, `tests/timing.rs`. 명시적으로 허용한 모듈은 `crates/crypto/src/sha256/`(x86-64와 AArch64), `crates/crypto/src/ct.rs`, `cmd/sibuna/src/signals.rs`, `cmd/sibuna-pow-wasm/src/lib.rs`, `cmd/console-ui/src/abi.rs`, `crates/store/src/ring.rs`입니다. 워크스페이스는 `unsafe_code = "deny"`와 Clippy의 `undocumented_unsafe_blocks = "deny"`를 설정합니다.

규율과 테스트. 함수 본문은 논리적 줄 70줄 이하, 파일은 1,408줄 이하이며, 검사 도구가 이를 강제합니다. 라이브러리 코드는 구조화된 오류를 반환하고 결코 직접 출력하지 않으며, 사용자가 보는 모든 오류는 위치를 밝히고, 발췌를 보여 주고, 힌트를 줍니다. 1.0.0-beta.3의 워크스페이스 테스트 모음은 SQLite에서 약 2,200개를 실행하고, PostgreSQL에서 약 700개를 더 실행합니다. 콘솔 인터페이스에는 골든 픽스처를 포함해 테스트 298개가 있으며, 브라우저 검사는 헤드리스 Chromium으로 중간 페이지와 모든 콘솔 페이지를 거쳐 갑니다[30].

11 한계와 남은 과제

어떤 주장도 근거보다 넓게 읽히지 않도록, 이 백서가 입증하지 않는 것을 나열합니다.

- 측정한 비교 대상은 하나. Anubis는 호스트 한 대에서 측정했습니다(Anubis와의 측정 비교). SafeLine과 호스팅형 WAF는 측정하지 않았습니다.
- 부하 상태의 클러스터 측정 없음. 기능 테스트는 PostgreSQL 데이터베이스 하나에서 노드 세 개를 실행하지만(SID 0011, Committed), 여러 노드에 부하를 거는 하니스는 없습니다.
- *PostgreSQL*은 소스 빌드 필요. 릴리스 실행 파일에는 SQLite 저장소와 콘솔이 있지만 PostgreSQL 지원은 없습니다. 클러스터에서는 `postgres-tls` 기능을 켜고 Sibuna를 빌드합니다.
- 콘솔의 격리 관문은 아직 실행하지 않았습니다. 부하 아래에서 활성 대시보드 8개로 측정하는 관문입니다(콘솔).
- 루프백만 측정. TLS, 광역 네트워크, 부하 아래의 실제 브라우저를 다룬 측정은 없습니다. 배포에서는 자기 경로를 직접 측정해야 합니다.
- 상한이 있는 검사. Shield는 본문의 처음 8 KiB를 보는 휴리스틱 검사기이며, 완전한 SQL 파서나 HTML 파서가 아닙니다. forward-auth 검사는 인그레스가 보내는 바이트만 봅니다.
- 프로토콜 범위. Sibuna는 HTTP/1.0과 HTTP/1.1을 지원하며, HTTP/2와 인바운드 TLS 종료는 지원하지 않습니다.
- *TLS* 지문은 인그레스에서 받습니다. Sibuna는 JA4를 직접 계산할 수 없습니다. 지문은 그것을 쓰는 인그레스만큼만 믿을 수 있으며, 그 인그레스는 클라이언트가 보낸 사본을 반드시 덮어써야 합니다. 클라이언트는 특화된 라이브러리로 브라우저의 핸드셰이크를 흉내 낼 수 있으므로, JA4는 자동화의 비용을 높일 뿐 신원을 증명하지는 않습니다[11].
- 암호학적 보증. 순차 작업 계층은 랜덤 오라클 모델에서 발표된 증명에 기댑니다. 우리의 구체적인 프로토콜에는 참조 라벨러와 대조하는 테스트가 있을 뿐, 독립적인 암호학 감사는 없습니다.
- 머신 한 대. 모든 수치는 AMD Ryzen 9 5950X 한 대에서 얻었습니다. 이 호스트에서의 순위가 다른 호스트에서의 순위는 아닙니다.

12 결론

웹 방어는 흔히 덧붙이는 방식으로 자라 왔습니다. 인터프리터 런타임, 컨테이너, 정규 표현식 규칙 집합, 외부 저장소가 애플리케이션 앞에 겹겹이 쌓였습니다. Sibuna에서 우리는 몇 가지 단순한 아이디어를 길잡이로 삼아 반대 방향을 시도했습니다.

- 송신자가 검증 가능한 작업으로 요청 허용의 대가를 치르게 하고, 서버의 검사는 싸고 상태 없이 유지하며, 세션이 자신이 치른 작업량을 기억하게 합니다.
- 요청 허용과 같은 패스에서 구조에 따라 선형 시간에 검사해, 어떤 세션도 검사를 우회할 수 없게 합니다.
- 머신을 존중합니다. 연결당 버퍼 하나, 할당 없는 요청 경로, 키 기반 배치를 쓰는 크기 제한 테이블, 새로 만들지 않고 재사용하는 스레드가 그 방법입니다.
- 성숙한 데이터베이스가 영속 공유 상태를 맡게 하고, 그 앞에는 요청 경로가 결코 기다리지 않는 저장소 스레드 하나를 둡니다.

16코어 머신 한 대에서 이 선택들 덕분에 Sibuna는 CPU 네 개로 forward-auth 모드에서 초당 200,000개가 넘는 허용 요청을 처리하며, 그 부하에서 상주 메모리는 약 16 MiB이고, 초당 100,000 요청에서 p99.9는 62 us입니다. 이는 루프백으로 측정한 호스트 한 대의 수치이며, 순위가 아니라 설계가 일관되게 맞물려 동작한다는 근거로 제시합니다. 우리는 자체 합의 프로토콜을 제거했고, 그 사실을 분명히 밝힙니다. 그 대가로 시스템은 더 작아졌고 데이터 평면에는 타사 의존성이 없습니다. 측정할 것이 아직 많이 남아 있으며, 그것이 정확히 무엇인지 밝히려고 노력했습니다.

13 출처와 재현

이 백서의 모든 수치는 Sibuna 저장소에 커밋된 결과 파일, Sibuna Discussion(SID) 기록, 또는 Sibuna 기술 안내서에서 왔으며, 각 표나 각주에 출처가 있습니다. 결과 파일은 `cargo xtask bench primitives`, `cargo xtask bench product`, `cargo xtask bench latency`가 기록하며, `benchmarks/results/README.md`에 각 파일의 명령, CPU 집합, 출처가 정리되어 있습니다.

- `benchmarks/results/primitives-latest.md`, `product-latest.md`, `latency-latest.md`와 각각의 `.json` 파일, `console-8-dashboards-2026-10-15.md`.

[30] SID 0010, "Acceptance criteria and implementation status" (기준 1, 3, 5). 이 기록은 SID 0014 이전에 워크스페이스 테스트 약 2,100개와 인터페이스 테스트 290개를 기록했습니다. 1.0.0-beta.3의 개수는 `cargo test --workspace -- --list`에서 얻었습니다.

- docs/sid/records/에 있는 SID 0002(기반 아키텍처와 개정 노트), SID 0004(Shield 운영 구성), SID 0006(수학적 기초), SID 0007(콘솔), SID 0010(워크스페이스, 크레이트, 의존성 정책), SID 0011(sqlx를 통한 저장소), SID 0012(콘솔 TLS), SID 0013(콘솔 저장소), SID 0014(인그레스에서 받는 JA4 지문).
- Sibuna 기술 안내서, docs/book/: 제I부(요청의 비용), 제II부(선행 연구), 제III부(암호학), 제IV부(프로토콜), 제V부(무할당 엔지니어링), 제VI부(오토마톤과 의미 기반 방화벽), 제VIII부(실측 평가), 제IX부(운영).
- 1차 문헌: Back, Hashcash (1997); Cohen and Pietrzak, Simple Proofs of Sequential Work (EUROCRYPT 2018); Blocki, Lee and Zhou, On the Security of Proofs of Sequential Work in a Post-Quantum World (ITC 2021); Aho and Corasick, Efficient String Matching (1975); Celis, Robin Hood Hashing (1986); ATM Forum의 Generic Cell Rate Algorithm (1996).

Sibuna 백서 | 2026년 11월 | 릴리스 1.0.0-beta.3
명세, 소스 코드, 벤치마크: https://github.com/insanai/sibuna