PURPLE WOLF: A Technical White Paper on a Low-Latency Traefik Web Application Firewall and Audit Relay

Victor Bona
Independent Researcher
Software Engineer
Blumenau, Brazil
victor.bona@hotmail.com

***Abstract*—PURPLE WOLF is a Rust implementation of a small, explicit, production-oriented Web Application Firewall for Traefik. It runs as a `wasm32-wasip1` http-wasm middleware, evaluates requests with a pure core engine, emits structured audit events, and ships a separate relay for signed webhook fan-out. This paper documents the product as an engineering artifact rather than as a marketing claim. We describe the threat model, request normalization path, detector groups, policy semantics, Traefik host boundary, relay protocol, release packaging, and operational limits. We then compare the design with ModSecurity, OWASP CRS, and Coraza, with particular emphasis on a committed head-to-head benchmark against the Coraza http-wasm Traefik plugin. Under the same Kubernetes topology, Traefik version, backend, node, and 200 m CPU plus 1 GiB memory budget, PURPLE WOLF added approximately 0.1 to 0.2 ms p99 over Traefik-only baseline, sustained clean service through 4000 RPS and mostly served 8000 RPS, while the measured Coraza http-wasm path collapsed at 500 RPS. Across 4536 OWASP CRS regression vectors spanning twelve classes, PURPLE WOLF blocked 660 vectors, or 14.55 percent, while Coraza inline Paranoia Level 1 rules blocked 277, or 6.11 percent. These results are useful but bounded: the corpus is a regression suite, not a real attack distribution, the benign set has only 53 requests, and Coraza native integrations are outside the measured scope.**

***Index Terms*—Web Application Firewall, Traefik, WebAssembly, http-wasm, Rust, Coraza, OWASP CRS, libinjection, Kubernetes, security benchmarking.**

I. INTRODUCTION

Web Application Firewalls still occupy a practical space between application code and perimeter infrastructure. They are not a substitute for secure application design, but they can reject common exploit traffic before it reaches a service, provide audit trails for attempted abuse, and give operators a deployable control point when code cannot be patched immediately. The open-source ecosystem around this role has historically centered on ModSecurity and the OWASP Core Rule Set, with Coraza providing a modern Go engine that supports ModSecurity SecLang and CRS compatibility [3], [4].

PURPLE WOLF takes a narrower position. It is not a general SecLang interpreter and does not attempt to load the full CRS rule set. It is a purpose-built Traefik middleware with high-precision detectors, simple policy modes, bounded state, and a relay protocol for audit delivery. The project is organized as a Rust workspace with three crates: `purple-wolf-core`, `purple-wolf-traefik`, and `purple-wolf-relay`. This separation is the central architectural decision. The core engine owns request modeling, normalization, detectors, policy, and audit records. The Traefik crate owns the http-wasm adapter and host interaction. The relay crate owns asynchronous webhook fan-out, signing, retry, dead-letter behavior, and metrics [1].

The contribution of this paper is a complete technical description of that product. It documents what PURPLE WOLF does, what it does not do, how it compares with adjacent WAF systems, and what its benchmark data actually supports. The intended reader is an engineer evaluating the tool, extending it, or deciding where it fits in a defense-in-depth architecture.

II. DESIGN GOALS AND THREAT MODEL

PURPLE WOLF is designed around five engineering goals. First, the inspection engine must remain pure and testable, without direct networking, async runtime coupling, or host-specific behavior. Second, the Traefik adapter must keep the host boundary explicit, because http-wasm is an ABI, not a Rust-native framework. Third, detector behavior must be explainable enough that an operator can understand why a request was blocked. Fourth, runtime cost must remain small relative to the proxy path. Fifth, audit delivery must be decoupled from request handling so slow subscribers do not slow the WAF.

The threat model is correspondingly limited. PURPLE WOLF is intended to detect and block high-confidence web attack patterns in HTTP requests, especially injection, traversal, scanner, structural, and simple reputation signals. It does not claim full CRS coverage, behavioral anomaly detection, bot management, positive security modeling, response-body inspection, or application-specific authorization. It also does not claim that a miss is acceptable simply because the WAF is narrow. Known gaps are documented as product facts, not hidden as caveats.

Table I summarizes the design constraints that shape the implementation.

TABLE I
PRIMARY DESIGN CONSTRAINTS

Constraint	Engineering effect
Pure engine	Core inspection logic can be unit-tested, property-tested, and fuzzed without Traefik or Kubernetes.
Explicit host boundary	The adapter reads config, headers, path, query, body, and peer metadata through http-wasm imports and translates results to host responses.
Precision first	Detector groups favor low false positives over broad CRS-style token recall.
Bounded state	Reputation tracking uses a bounded LRU token bucket keyed by source IP.
Async relay isolation	Audit fan-out runs outside the request path, with per-subscriber queues and drop accounting.

TABLE II
WORKSPACE COMPONENTS

Crate	Responsibility
`purple-wolf-core`	Builds normalized request models, runs detector groups, resolves policy decisions, and emits scrubbed audit entries.
`purple-wolf-traefik`	Implements the http-wasm guest, parses Traefik plugin config, reads request fields, applies body-cap behavior, writes blocks, and logs audits.
`purple-wolf-relay`	Parses audit logs, enriches envelopes, signs webhook payloads, handles retry and dead-letter delivery, and exposes health and metrics.

III. WORKSPACE ARCHITECTURE

The repository is a Rust 2021 workspace. The core crate, `purple-wolf-core`, contains the inspection engine. Its modules include request construction, config, detectors, policy, audit serialization, clock abstraction, and the libinjection foreign-function boundary. The adapter crate, `purple-wolf-traefik`, compiles to `wasm32-wasip1` and exports the http-wasm request handler used by Traefik. The relay crate, `purple-wolf-relay`, is a standalone Tokio service that reads audit events from configured sources, transforms them into a stable envelope, and delivers them to subscribers.

This organization avoids a common failure mode in edge security middleware: mixing detector semantics with proxy plumbing. In PURPLE WOLF, detector code does not know that Traefik exists. The adapter does not decide detector meaning. The relay does not reclassify WAF decisions. Each boundary is narrow enough that an operator can map a live event to its source.

IV. REQUEST MODEL AND NORMALIZATION

The core request model is built by `Request::build`. It normalizes method, host, path, query, headers, source IP, and body into an immutable request structure for detector evaluation. Query strings are percent-decoded for inspection, while the raw query is preserved for audit. Body bytes are stored as raw bytes. Header inspection is intentionally selective: `cookie`, `referer`, `host`, `authorization`, `user-agent`, and any `x-*` header are considered inspectable. Both raw and percent-decoded forms are inspected where relevant.

The selectivity matters. Full header inspection can create high false-positive rates because application headers often carry arbitrary encodings, bearer tokens, tracing identifiers, and opaque values. PURPLE WOLF instead picks headers that are commonly attack-bearing or operator-controlled. The `x-*` rule gives teams room to inspect custom ingress headers without enumerating every possible name.

Audit records include host, path, raw query, method, source IP, action, blocked rule, blocked severity, blocked detail, would-block rules, and optional labels. Control characters are scrubbed before emission, which prevents audit logs from becoming a terminal-control or log-injection channel.

V. DETECTOR GROUPS

PURPLE WOLF has four main detector families: injection, signatures, structural, and reputation. Each detector returns rule findings with severity and details. The policy layer later determines whether those findings are enforced or recorded as would-block evidence.

A. Injection

The injection detector uses vendored libinjection through a small raw-byte FFI wrapper. libinjection describes itself as a SQL and SQLi tokenizer, parser, and analyzer, with APIs for detecting SQL injection and XSS-style payloads [8]. PURPLE WOLF delegates SQLi and XSS recognition to that engine, but it owns where and how inputs are passed into it. This is important because a WAF can have strong payload classifiers and still miss attacks if it inspects the wrong request fields or normalizes them inconsistently.

B. Signatures

The signature detector uses case-insensitive Aho-Corasick matching over static literals. The current literal set includes traversal and shell markers such as `../`, `..\`, `textbackslash`, `/etc/passwd`, `$(`, backtick, `/bin/sh`, and scanner markers such as `sqlmap`, `nikto`, and `nuclei`. The intent is not to recreate CRS. It is to cover high-confidence strings that have low benign probability and high operational value.

C. Structural

The structural detector validates request shape. It checks allowed methods, a 16 KiB aggregate header-byte cap, and a 100-header count cap. Structural enforcement is configuration-controlled. In the benchmark configuration discussed later, only

TABLE III
DETECTOR GROUP SEMANTICS

Group	Inputs	Typical findings
Injection	Path, query, selected headers, body bytes under cap	SQLi and XSS findings from libinjection.
Signatures	Path, query, selected headers, body bytes under cap	Traversal, shell markers, scanner identifiers, sensitive-path literals.
Structural	Method and header shape	Disallowed method, excessive header bytes, excessive header count.
Reputation	Source IP plus configured deny list	Rate or deny-list decisions with bounded source tracking.

the injection and signatures groups were enabled, which is why TRACE and CONNECT robustness probes were not blocked.

D. Reputation

The reputation detector keeps bounded source-IP state using a token bucket and LRU eviction, with a default cap of 50,000 tracked IPs. It can also deny listed IPs. The design goal is bounded memory under abusive source cardinality. It is not a behavioral bot classifier.

VI. POLICY SEMANTICS

Policy is separate from detection. A detector can find a rule hit without necessarily blocking the request. The policy layer evaluates global mode and per-group mode. If both global mode and the detector group are in enforce mode, the highest-severity enforced verdict becomes the blocking rule. Other findings are retained as would-block evidence. If a group is in monitor mode, findings are emitted for audit without blocking.

This model supports progressive rollout. Operators can deploy the middleware in monitor mode, observe would-block rules, then move individual groups to enforce mode. It also avoids losing evidence when one rule blocks the request first. The audit record retains additional candidate findings so downstream analysis can see the complete security context.

VII. TRAEFIK AND HTTP-WASM INTEGRATION

Traefik supports plugin systems for extending the proxy. Its documentation states that WebAssembly plugins can be developed in any language that compiles to Wasm and are based on http-wasm [7]. The http-wasm ABI defines an HTTP guest that exports `handle_request` and `handle_response`; the host calls these functions and exposes request fields through imported functions [6].

PURPLE WOLF implements the guest side directly in Rust. The adapter reads Traefik plugin configuration, initializes the core engine, pulls request fields from host imports, and returns either a block response or `next=1` to continue to the next handler. The request path is wrapped in `catch_unwind`; when adapter or detector code fails unexpectedly, `failMode` controls whether the middleware fails open or closed.

The adapter has two details worth emphasizing. First, invalid Middleware configuration does not silently disable protection. It falls back to all detector groups enabled in monitor mode. This makes misconfiguration visible without creating an accidental hard block. Second, request body inspection is capped by `maxInspectBytes`. If the body exceeds the cap, `overCap=block` rejects the request and `overCap=pass` skips body inspection. This keeps request memory bounded and makes oversized-body behavior explicit.

VIII. WEBHOOK RELAY

The relay is a separate service, not a function inside the WAF middleware. That split protects the request path from subscriber slowness, back pressure, and third-party webhook instability. Relay configuration uses strict deserialization with unknown fields denied. At least one source must be configured. Zero subscribers are allowed, which lets packaged deployments start safely and remain idle until an operator adds destinations.

The pipeline has four stages. Source tasks read audit events. A parser and enrich stage strips ANSI control sequences, extracts the first balanced JSON object, validates the audit shape, and builds a stable `purple-wolf.audit/v1` envelope. A fan-out stage sends events to subscriber queues without blocking globally. Subscriber tasks deliver events over HTTP with per-subscriber retry and dead-letter behavior.

Webhook signing uses HMAC-SHA256 over `timestamp.raw_body`. Signatures are emitted as `sha256=<hex>`. HTTP delivery classifies 2xx as delivered, 408, 429, 5xx, network errors, and timeouts as retryable, redirects disabled as a permanent class, and other 4xx responses as permanent. The admin surface exposes `/healthz`, `/readyz`, `/metrics`, and `/version`. In v0.3 this surface is intended to be internal-only and has no built-in authentication.

IX. DEPLOYMENT AND RELEASE SURFACES

PURPLE WOLF ships multiple operational surfaces: Traefik Middleware examples, a relay, Docker-oriented examples, Kubernetes manifests, Helm chart packaging, Kustomize deployment overlays, signed release artifacts, SBOM generation, and release verification documentation. The CI contract includes workspace tests, doctests, formatting, clippy with warnings denied, supply-chain and license checks with `cargo deny`, a wasm release build for the Traefik adapter, fuzz smoke tests, coverage checks, docs checks, and Docker integration suites for both Traefik and relay paths.

The release posture is important for WAF software. A WAF that is technically correct but difficult to verify becomes operationally risky. PURPLE WOLF therefore treats packaging evidence as part of the product. SBOMs, signed WASM and relay artifacts, Helm OCI publication, and Kubernetes examples are not ancillary docs. They are part of the trust chain for users deciding whether to run a request-filtering component in front of their applications.

X. Related Systems

ModSecurity is the conventional open-source WAF engine lineage. The OWASP CRS project describes itself as a set of generic attack detection rules for ModSecurity or compatible WAFs, covering categories such as SQL injection, cross-site scripting, and local file inclusion with a goal of minimal false alerts [3]. Coraza is a Go WAF engine that supports ModSecurity SecLang and claims compatibility with OWASP CRS [4]. These systems are broader than PURPLE WOLF in rule-language support and generic rule coverage.

PURPLE WOLF is closer to a specialized middleware than to a generic WAF platform. It does not interpret SecLang. It does not ship CRS. It does not aim to be a drop-in replacement for Coraza or ModSecurity. Its competitive claim is narrower: for Traefik users who want a small Rust http-wasm middleware with bounded memory, explainable detectors, and audit relay integration, the specialized design can be easier to reason about and significantly cheaper at runtime than a full rule-engine path.

The benchmark comparison in this paper is therefore carefully scoped. It compares PURPLE WOLF v0.3 against the Coraza http-wasm Traefik plugin v0.3.0 under one Kubernetes topology and one inline Paranoia Level 1 style ruleset. The Coraza http-wasm Traefik repository itself warns that the project is early-stage and recommends native Coraza integration with Traefik for production-grade performance [5]. The results should be read as a wasm-vs-wasm comparison, not as a verdict on Coraza native integrations.

XI. Benchmark Methodology

The benchmark corpus and runners are committed under `benchmarks/`. The round-2 benchmark used a same-node Kubernetes topology: Traefik v3.1 plus `whoami` backend for PURPLE WOLF, Traefik v3.1 plus the same backend for Coraza, and a Traefik-only baseline pod for latency floor measurement. All pods ran on the same K3s 1.30 node with the same resource envelope: 200 m CPU request and 1 GiB memory limit. The benchmark used the same backend, node, Traefik version, target mix, and runner shape for comparable measurements [2].

For detection, the expanded corpus replayed 4536 OWASP CRS regression vectors across twelve classes. A small benign set of 53 hand-curated requests was used for false-positive observation. For load, vegeta used a ten-target mix: seven benign GETs and three attacks, with 403 responses counted as non-success by vegeta even when they represent correct WAF blocking. This means a healthy WAF at low load shows approximately 0.70 success in the mixed workload.

XII. Detection Results Against Coraza

Table V is the most important comparative table in the paper. It is computed from committed round-2 CSV outputs. PURPLE WOLF blocked 660 of 4536 vectors. Coraza blocked 277 of the same 4536 vectors. Both WAFs showed 0 percent false-positive rate on the 53-request benign corpus, but the benign sample is too small for a broad false-positive claim.

TABLE IV
BENCHMARK CONFIGURATION

Axis	Value
Proxy	Traefik v3.1
Backend	`traefik/whoami:v1.10`
Cluster	K3s 1.30 on a single NixOS node
Resource budget	200 m CPU request, 1 GiB memory limit
PURPLE WOLF config	`mode=enforce`; injection and signatures enforce; body cap 1 MiB; fail-open
Coraza config	`SecRuleEngine On`; request body access; inline SQLi, XSS, scanner, traversal, and method rules
Attack corpus	4536 CRS regression vectors across twelve classes
Benign corpus	53 hand-curated clean requests

The strongest PURPLE WOLF margins were Java attack vectors, RCE, XSS, LFI, and generic attack classes. Coraza was stronger on the seven-vector scanner class and slightly stronger on SQLi. These class-level differences matter more than the aggregate. The overall 2.38x block-count ratio is useful, but it can hide the fact that both systems still have modest recall on CRS atomic-token regression tests. PURPLE WOLF is stronger in this measured configuration, but neither system should be described as complete attack coverage.

XIII. Latency, Throughput, and Soak Results

The no-WAF baseline measured the Traefik floor. At 100, 500, and 1000 RPS, baseline p99 was approximately 1.0 ms, 0.8 to 1.1 ms, and 0.7 ms. PURPLE WOLF added approximately 0.1 to 0.2 ms p99 over that floor at the same low-to-moderate rates. The sustained soak at 1000 RPS ran for 600,000 requests over ten minutes, with p50 0.446 ms, p95 0.714 ms, p99 0.838 ms, and stable 0.70 vegeta success, which corresponds to seven benign targets in a ten-target mixed workload.

Table VI shows that the observed break point is between 8000 and 12000 RPS under the stated resource budget and shared-node topology. The 8000 RPS result is mixed: one iteration remained healthy, while another showed tail stress and reduced success. The 12000 and 16000 RPS steps collapsed. The benchmark notes that the shared K3s control plane became briefly unstable at the high-rate steps, so these values should be interpreted as a ceiling on this node, not as a universal property of the WAF.

The runtime comparison is where the specialized design is clearest. In the same benchmark document, the Coraza http-wasm path degraded at 500 RPS, with a second 500 RPS iteration showing p50 around 10 seconds and p99 around 28 seconds, then timeout-floor behavior at 1000 RPS. This does not mean Coraza as an engine is slow in all integrations. It means the measured http-wasm Traefik path behaved poorly under this specific resource envelope and inline ruleset.

XIV. Robustness Findings

The benchmark also included targeted probes. Header-borne SQLi in `Cookie`, XSS in `Referer`, and SQLi in an

TABLE V
ROUND-2 DETECTION RESULTS ACROSS CRS CLASSES

CRS class	Vectors	PURPLE WOLF blocked	PURPLE WOLF TPR	Coraza blocked	Coraza TPR
Scanner detection	7	1	14.29%	3	42.86%
Protocol enforcement	424	4	0.94%	3	0.71%
Protocol attack	112	2	1.79%	1	0.89%
Local file inclusion	75	13	17.33%	10	13.33%
Remote file inclusion	41	3	7.32%	3	7.32%
Remote command execution	876	87	9.93%	32	3.65%
PHP injection	399	25	6.27%	21	5.26%
Generic attack	234	8	3.42%	3	1.28%
Cross-site scripting	217	62	28.57%	51	23.50%
SQL injection	934	121	12.96%	127	13.60%
Session fixation	44	1	2.27%	1	2.27%
Java attack	1173	333	28.39%	22	1.88%
Overall	4536	660	14.55%	277	6.11%

TABLE VI
PURPLE WOLF RAMP-TO-BREAK RESULTS

Target RPS	p50 ms	p95 ms	p99 ms	Success
2000	0.44	0.69	0.89	0.70
4000	0.47	0.75	1.28 to 3.14	0.70
8000	0.61 to 0.65	0.95 to 1.01	3.26 to 79.80	0.65 to 0.70
12000	0.12 to 0.39	2.84 to 82.44	13616 to 27374	0.13 to 0.33
16000	0.10	13.02 to 231.71	about 30000	about 0.01

TABLE VII
SELECTED RUNTIME STABILITY RESULTS

Measurement	Result
Isolated PURPLE WOLF p99 overhead	Approximately 0.1 to 0.2 ms over Traefik-only baseline.
PURPLE WOLF 10-minute soak	1000 RPS, 600,000 requests, p99 0.838 ms, success 0.70.
PURPLE WOLF soak CPU	270 m median, 285 m maximum in committed samples.
PURPLE WOLF soak memory	Warmed band approximately 86 to 96 MiB, with committed samples ranging 36 to 106 MiB.
Coraza http-wasm load	Degraded at 500 RPS and reached timeout-floor behavior at 1000 RPS in round-1 load results.
Coraza memory note	Round-1 resource samples reached 946 MiB maximum under the same 1 GiB limit.

`X-*` custom header were blocked. Benign cookie traffic was allowed. Direct path traversal, query traversal, `$(whoami)`, and `/bin/sh` were blocked. However, a User-Agent SQLi payload prefixed with `Mozilla/5.0` was missed, and a bare `;wget evil.com/x` query payload was missed. TRACE and CONNECT were not blocked in the benchmark because the structural group was not enabled.

These findings are valuable because they prevent the paper from overstating the product. The current detector set is high precision and fast, but it leaves gaps in context-sensitive User-Agent injection and some shell command forms. An operator requiring broad command-injection coverage should add custom signatures or layer PURPLE WOLF with another WAF stage. An operator requiring unusual method blocking should enable structural enforcement.

XV. SCIENTIFIC LIMITS OF THE BENCHMARK

The benchmark is useful because the source code, manifests, corpora, runners, and raw results are committed. It is limited because it is a single-node, single-pod, single-backend experiment. The attack corpus is the OWASP CRS regression corpus, not a distribution of live attacks. The false-positive corpus has only 53 hand-curated benign requests. The Coraza configuration uses inline Paranoia Level 1 style directives rather than full CRS includes, because reliable host file-system mapping for full CRS was not available in the measured http-wasm path. The Coraza wasm plugin also documents its early-stage status and points production performance users toward native integration [5].

A stronger study would add a one-hour or one-day soak, a captured benign traffic trace, full CRS where host mounting is

reliable, native Coraza integration as a separate comparison arm, and repeated runs on a dedicated benchmark node. The current evidence supports a narrower conclusion: in this committed benchmark, PURPLE WOLF is materially faster than the Coraza http-wasm Traefik path and blocks more CRS regression vectors under the same measured inline-rule configuration.

XVI. OPERATIONAL GUIDANCE

For production use, PURPLE WOLF should be rolled out in stages. Start in monitor mode, inspect audit entries and would-block findings, then enforce one detector group at a time. Enable structural enforcement if method and header-shape blocking are desired. Treat the relay admin surface as internal-only. Use HMAC verification on subscribers. Keep body caps explicit, especially for services accepting uploads. If the relay has no subscribers during initial deployment, that is a valid safe state, not a failed install.

The most important operational rule is to avoid confusing WAF coverage with application security coverage. PURPLE WOLF can reduce exposure to common exploit payloads and create a useful audit stream, but it does not validate authorization, business logic, schema-level input contracts, or authentication flows. It should sit beside secure coding, dependency management, rate limiting, authentication, and observability.

XVII. CONCLUSION

PURPLE WOLF is a deliberately scoped WAF and audit system for Traefik. Its technical value comes from separation of concerns: a pure Rust inspection core, an explicit http-wasm adapter, policy semantics that support monitor and enforce modes, bounded reputation state, and an out-of-path webhook relay. The benchmark evidence shows very low p99 overhead, stable ten-minute behavior at 1000 RPS, a practical single-pod ceiling around 8000 RPS on the measured shared node, and stronger CRS regression-vector blocking than the Coraza http-wasm Traefik plugin under the same benchmark conditions.

The product is not a replacement for ModSecurity, Coraza native integrations, or full OWASP CRS when those systems are required. It is a smaller and faster design for teams that want a precise Traefik-native edge guard with inspectable behavior and signed audit delivery. Future work should expand the benign corpus, complete longer soak testing, add structured custom signatures, characterize missed CRS classes more deeply, and compare against native Coraza in a separate benchmark arm.

REFERENCES

[1] Guara Cloud, "purple-wolf repository," GitHub. [Online]. Available: https://github.com/guaracloud/purple-wolf

[2] Guara Cloud, "purple-wolf vs Coraza: head-to-head WAF benchmark," `docs/benchmark.md` and `benchmarks/results/round2-20260528-233221/`, 2026. [Online]. Available: https://github.com/guaracloud/purple-wolf

[3] OWASP Foundation, "OWASP CRS," 2026. [Online]. Available: https://owasp.org/www-project-modsecurity-core-rule-set/

[4] OWASP Coraza, "Coraza - Web Application Firewall," 2026. [Online]. Available: https://www.coraza.io/docs/tutorials/introduction/

[5] J. C. Chavez, "Coraza http-wasm Traefik plugin," GitHub, 2024. [Online]. Available: https://github.com/jcchavezs/coraza-http-wasm-traefik

[6] http-wasm project, "HTTP Handler Application Binary Interface (ABI)," 2026. [Online]. Available: https://http-wasm.io/http-handler-abi/

[7] Traefik Labs, "Extend Traefik," 2026. [Online]. Available: https://doc.traefik.io/traefik/extend/extend-traefik/

[8] libinjection project, "SQL / SQLI tokenizer parser analyzer," GitHub, 2026. [Online]. Available: https://github.com/libinjection/libinjection