Lateralus Pentester: Technical Architecture

Backend services, data model, and the pipeline execution engine

Lateralus Language

bad-antics · April 2026 · Lateralus / nullsec

ABSTRACT This paper describes the internal architecture of the Lateralus Pentester backend: the microservices, the shared data model, the pipeline execution engine, and the evidence storage system. It is intended for self-hosting operators and security researchers who want to understand the platform's internals for audit purposes.

1. Service Architecture

Lateralus Pentester is a microservices application with five core services:

```
Service                Port    Responsibility
-----------------------------------------------
api-gateway            443     TLS termination, auth, rate limiting
engagement-service     8001    Engagement CRUD, finding storage
pipeline-executor      8002    Run nullsec pipelines, stream results
evidence-store         8003    Evidence upload, storage, signing
report-generator       8004    Template rendering, PDF export
```

All services communicate via gRPC with mutual TLS authentication. The api-gateway is the only service exposed to the internet; all other services are internal to the Kubernetes cluster.

2. Data Model

The shared data model is defined in Lateralus types and exported to the other services via gRPC protocol buffers (generated from the Lateralus types using the ltl proto tool):

```
// Core data model
record Engagement { id, client, scope, timeline, findings, team }
record Finding    { id, title, severity, evidence, status, cvss }
record Evidence   { id, kind, content_hash, timestamp, signer }
record Pipeline   { id, definition, last_run, results }
```

The Evidence.content_hash is the SHA-256 of the evidence content, linked to the evidence store. The Evidence.signer is the Ed25519 signature over the hash, signed by the user who uploaded the evidence.

3. Pipeline Execution Engine

The pipeline-executor service receives a pipeline definition (a serialized Lateralus pipeline expression) and an engagement scope, compiles the pipeline to LBC bytecode, and executes it in a sandboxed environment.

```
// Execution flow
let request: PipelineRunRequest = recv_from_api();
let lbc = ltl::compile(request.pipeline_definition)?;
let sandbox = Sandbox::new({
    scope:       request.scope,
    network_cap: NetworkCapability::InScope(request.scope),
    timeout:     request.timeout,
});
let results = sandbox.run(lbc) |>> stream_to_client;
```

The sandbox uses Linux namespaces and seccomp to restrict the network access of the pipeline to the engagement scope. A pipeline that attempts to scan an out-of-scope host receives a connection refused from the sandbox's network filter.

4. Evidence Store

The evidence store is a content-addressed storage system. Evidence is stored by its SHA-256 hash; the hash is the canonical identifier. Duplicate evidence (same hash) is stored once.

```
// Evidence upload
fn store_evidence(content: &[u8], signer: Ed25519PrivKey) -> EvidenceId {
    let hash = sha256(content);
    let sig = ed25519_sign(&signer, &hash);
    storage::put(hash, content);   // no-op if already exists
    evidence_db::record(hash, sig, time::now());
    EvidenceId(hash)
}
```

Evidence is stored in an append-only manner: existing evidence is never modified or deleted (unless subject to legal retention policies). The storage backend is S3-compatible; self-hosting operators can use MinIO, AWS S3, or Backblaze B2.

5. Authentication and Authorization

The api-gateway authenticates all requests using JWTs from the configured OIDC provider. Authorization is RBAC with four roles:

```
Viewer:     read-only access to assigned engagements
Analyst:    create and edit findings, upload evidence
Lead:       manage engagement scope, assign team, run pipelines
Admin:      all actions, team management, integration config
```

Each service verifies the JWT and the role claim on every request. There is no session state in the services; all authorization information is in the JWT.

6. Audit Logging

All state-changing actions are recorded in an append-only audit log using the same hash-chained ledger mechanism as the element-115-drive telemetry system (paper 14). Each audit record is signed by the service's Ed25519 key and includes: the action, the actor, the target resource, and the timestamp.

The audit log is stored separately from the application database and is exposed read-only to the SIEM integration. It cannot be modified retroactively — only new records can be appended.

7. Disaster Recovery

The platform provides RPO < 1 hour and RTO < 4 hours via:

```
PostgreSQL: streaming replication + WAL archiving to S3
Evidence store: S3 cross-region replication
Application state: Kubernetes etcd snapshots every 15 minutes
Recovery: Terraform + Helm charts for infrastructure rebuild
```

Backup restoration is tested monthly in an isolated environment as part of the platform's operational runbook.

8. Monitoring and Observability

The platform uses OpenTelemetry for distributed tracing, Prometheus for metrics, and structured JSON logging for log aggregation. A Grafana dashboard ships with the self-hosting deployment:

```
Key metrics:
  pipeline_runs_total          # engagement activity
  pipeline_run_duration_ms     # tool performance
  findings_created_total       # engagement productivity
  api_request_duration_ms      # latency SLO
  evidence_store_bytes_total   # storage growth
```

The SLO targets: API p99 latency < 500ms, pipeline executor startup < 5s, evidence upload < 2s for files up to 50MB.