DM-Protect — MASTER SYSTEM OVERVIEW (complete, exhaustive, production-grade blueprint)

This is the **final, single-source master blueprint** for DM-Protect. It collects and unifies *every* feature, function, file, dataflow, model, policy, security control, deployment option and operational artifact we discussed — **no minimal shortcuts**, all enterprise-grade, full detail. Use this as your canonical engineering, security and product playbook.

Length: long. Skim using headings. Everything below is designed to be actionable and auditable. If you want this exported as a PDF or repo-skeleton next, tell me and I'll produce it.

Table of contents

17. Observability, KPIs, SLOs, monitoring and dashboards
18. Operations: runbooks, drills, incident lifecycle, staffing
19. Developer file map (complete repo layout with file purposes)
20. Exact prioritized build roadmap & day-by-day initial plan
21. Tests & verification suite (unit/integration/chaos/adversarial)
22. Appendix: sample schemas, playbooks, commands, owner scripts

1 — Vision, scope & guarantees

Vision: DM-Protect is the world's most advanced owner-bound defensive AI platform: an always-on, self-observing, self-healing, continuously-learning security brain that protects endpoints, networks, cloud, web apps, IoT/OT and physical sensors — while remaining auditable, tamper-resistant and owner-controlled.

Guaranteed behaviors (architectural guarantees):

- Local detection & enforcement work **fully offline**. Cloud availability is optional; cloud outages do **not** reduce protection capability.
- All sensitive operations (model promotion, code access, major remediation) require cryptographic owner approval (WebAuthn attestation / signed token) unless explicitly configured.
- All artifacts that could modify runtime (binaries, models, signed playbooks) are signed and verified before use.
- Self-healing follows a tiered, auditable, reversible process; owner can revert any change.
- No offensive/exploit capability is included; DM-TestSecure only simulates behavior in an air-gapped environment.

2 — System principles & non-negotiables

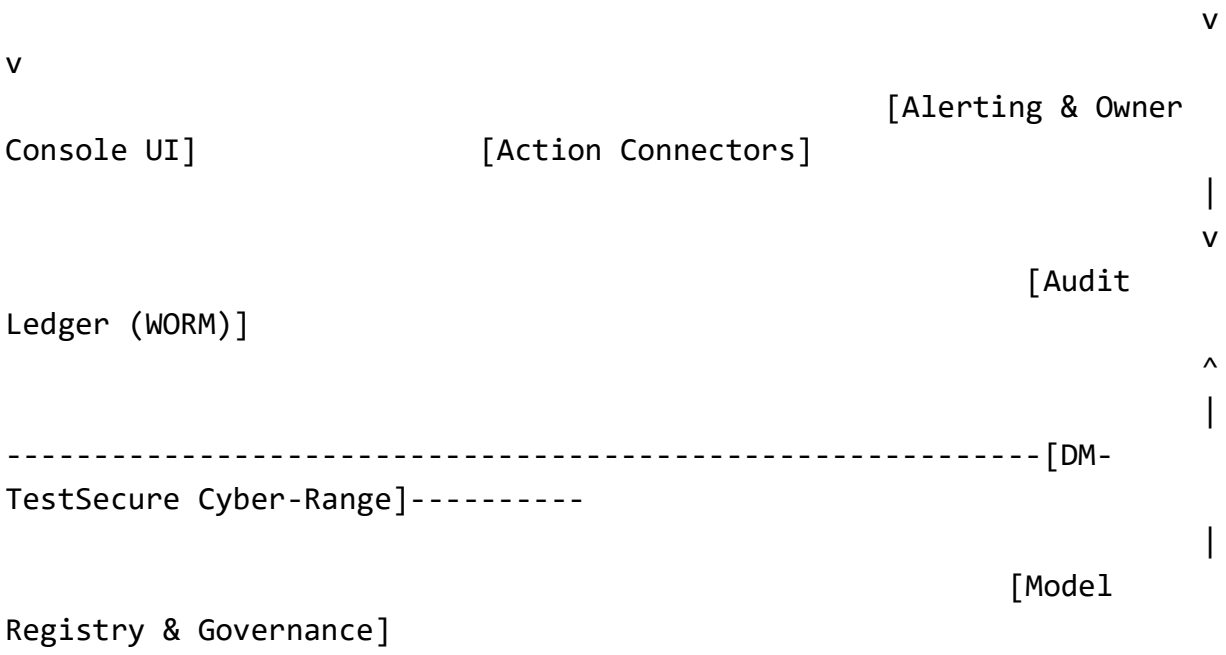
1. **Defensive-only:** No offensive exploit generation or instructions.
2. **Owner-first cryptographic control:** WebAuthn/FIDO2 + TPM/HSM-backed keys.

3. **Privacy-by-default:** local feature extraction, PII minimization, DP for cross-tenant learning.
4. **Auditable & tamper-evident:** chained ledger + WORM storage + external anchoring option.
5. **Human-in-the-loop for high-risk decisions:** owner or multi-signer required.
6. **Defense-in-depth:** multiple independent detectors (ensemble) + deception + self-heal.
7. **Reproducible & signed supply chain:** SBOM + cosign + two-party signing for production.
8. **Fail-safe & reversible:** all automated actions have rollback artifacts and verification steps.
9. **No single point of silence:** hybrid operation, offline mode, local-only capability.
10. **Legal & compliance aware:** DPA, legal hold, documented pen-test authorizations.

3 — High-level architecture & runtime modes

Architecture overview (text diagram)

```
[Agents & Sensors] --signed events--> [Collector / Normalizer / Enricher] --> [Telemetry Lake (S3)] & [Event DB (ClickHouse)]
                                                                                                    |
                                                                                                    v
                                                                                          [Feature Store / Cache (Redis)]
                                                                                                    |
                                                                                                    v
                                                                                  [Inference Cluster (ONNX/TorchServe / Ensembling + SHAP)]
                                                                                                    |
                                                                                                    v
                                                                                        [Decision Engine (policy)] --------> [SOAR / Orchestrator]
                                                                                                    |
                                                                                                    |
```



Runtime modes

- **Hybrid (recommended):** local inference + optional outbound for threat intel, model promotions, dashboarding and managed SOC. Use mTLS, pinned certs, scheduled update windows.
- **Offline / Air-gapped:** no outbound network; signed bundles via USB/HSM for updates; local-only telemetry & model operations.
- **Cloud/SaaS:** lightweight agents + cloud backend (for SMBs). Not recommended for regulated high-risk customers.

4 — Full component inventory (roles & responsibilities)

1. **Owner Control Plane:** enrollment, approvals, emergency keys, owner audit UI.
2. **Agents & Sensors:** endpoint agent (Go/Rust), mobile agent (Android/iOS), network sensors (Zeek/Suricata), cloud connectors, sidecars, physical sensor adapters.
3. **Collector / Normalizer / Enricher:** secure ingest, signature verification, canonical schema mapping, asset enrichment (CMDB/SBOM).

4. **Telemetry Lake & Event DB:** raw encrypted blobs (S3), indexed events (ClickHouse/Elasticsearch), time-series metrics (Timescale/Prometheus).
5. **Feature Store:** real-time features, sliding-window aggregates, cached for low-latency scoring.
6. **Inference Cluster:** multiple model families served (ONNX/TorchServe/custom), ensemble aggregator, explainability (SHAP).
7. **Decision Engine & Policy Manager:** policy DSL, scoring thresholds, action gating, owner-required logic.
8. **SOAR / Orchestrator:** execute playbooks, connectors to firewalls, cloud APIs, agent RPCs, transactional execution + rollback.
9. **Guardian Core:** signed minimal runtime on endpoints and appliances that enforces integrity and lockdown.
10. **DM-TestSecure:** cyber-range for simulation, labeled trace generation and adversarial testing.
11. **Model Registry & Governance:** model artifacts, provenance, evaluation metadata and model promotion tokens.
12. **Audit Ledger:** append-only hash chain, WORM storage, optional public anchoring.
13. **UI / Owner Console / SOC Workbench:** dashboards, approvals UI, model review, playbook editor, defense idea review.
14. **CI/CD & Signing Pipeline:** reproducible builds, SBOM generation, cosign signing, two-party release gating.
15. **Key Management:** HSM/Vault integration, TPM device binding, PQC-ready key options.
16. **Deception Fabric & Honeypots:** dynamic honeypots, capture & sanitized telemetry for training.
17. **DLP / Insider Module:** content fingerprinting, HR integration, JIT privileged access.
18. **Federated Learning Controller:** secure aggregation & DP for cross-tenant learning.
19. **Backup & Recovery:** signed recovery images, offline bundle ingestion, emergency seed management.
20. **Monitoring / Observability:** Prometheus/Grafana, model health exporters, alerting rules.

5 — End-to-end data flow & trust chain (step-by-step)

1. **Agent collects event** → builds canonical payload → computes `payload_hash` → signs payload using device key (TPM/HSM or software key for PoC) → sends to collector over mTLS.
2. **Collector validates** signature and hash, stores raw encrypted blob in S3 with metadata, normalizes into canonical event schema and inserts indexed row into Event DB.
3. **Feature extractor** consumes events and produces sliding-window features / embeddings; writes features to Redis cache and cold features to S3.
4. **Inference request**: Decision Engine or collector calls `/score` with features; inference service loads signed models, runs per-model scoring (baseline, sequence, graph, micro-timing) and returns scores + explainability.
5. **Decision Engine** aggregates scores, applies policy (ABAC/RBAC and owner rules), creates `Alert` or `ActionTicket`.
6. **SOAR / Orchestrator** executes action (e.g., isolate host) with pre-checks; records execution and rollback artifacts; writes action entry to Audit Ledger.
7. **Owner UI** receives notifications for major actions and displays evidence, SHAP explanations, and approval buttons (WebAuthn challenge).
8. **Model training loop**: DM-TestSecure + sanitized real telemetry produce training datasets; training runs in sandbox; model bundle created + SBOM + metrics; owner reviews and signs to promote to canary; canary monitors metrics then promotes to full.
9. **Audit & ledger**: every critical operation (owner approvals, model promotions, major remediation) is appended to the ledger with a signer and cryptographic chain; optional anchoring to external public ledger for extra tamper-evidence.

6 — Feature-by-feature specification (function → how → files → inputs/outputs → tests → security)

Below are every major feature we discussed. For each I list **what it does, how it's built (algorithms & components), exact files/modules to implement, inputs/outputs, tests you must write, and security/governance controls**. This is exhaustive and intended to be used as the engineering spec.

Note: file paths reference the canonical repo structure (see Section 19). If you want, I can produce each file content next.

A — Owner Enrollment & Approval Flow

Function: register owner device(s), bind attested WebAuthn keys, generate owner tokens for approvals and promotions.

How: WebAuthn/FIDO2 registration (direct attestation), store public key and attestation metadata (manufacturer certificate chain), implement challenge/response sign flow for single-use `approval_token` JWTs signed by owner key.

Files:

- `collector/api/enroll.py` — endpoints:
  - `POST /owner/enroll/start` → returns WebAuthn challenge
  - `POST /owner/enroll/complete` → verifies attestationObject and clientDataJSON; stores `owner_record`
- `decision-engine/approval/approval.go` — create challenge, verify signatures, mint `approval_token`
- `ui/web/src/pages/OwnerApproval.tsx` — UI for listing pending approvals and invoking sign flow
- `docs/OWNER_ONBOARDING.md` — step-by-step admin guide

Inputs: WebAuthn attestation objects, owner device metadata.

Outputs: `owner_record` in DB, `approval_token` (short-lived), audit ledger entry.

Tests:

- Enroll positive flow (valid attestation), negative (invalid chain), replayed signature test, expired challenge.

Security:

- Do not store raw biometrics. Enforce `userVerification: required` in WebAuthn. Use hardware-backed keys. Provide emergency key recovery with multi-sig HSM or physical seed.

B — Agent: Collection, Signing, Enforcement

Function: collect telemetry, sign payloads, provide local enforcement and attestation, self-protect.

How: agent in Go/Rust; eBPF for Linux syscall tracing; efficient process & socket tracking; local signing via TPM or software key; gRPC for orchestrator commands; watchdog & Guardian Core collaboration.

Files:

- `agent/cmd/agent/main.go` — main bootstrap
- `agent/pkg/collection/process_collector.go`
- `agent/pkg/collection/netflow_collector.go`
- `agent/pkg/signer/device_signer.go`
- `agent/pkg/enforce/isolate.go` — implements isolation via iptables/nft or hypervisor APIs
- `agent/config/agent.yaml`

Inputs: kernel/process events, sensor streams.

Outputs: signed event envelopes to `POST /telemetry/event`.

Tests: unit collectors, signing verification, agent restart logic, kill/disable detection by Guardian Core.

Security: sign with TPM/HSM; prevent silent disable; require Guardian Core intervention on tamper; agent image must be cosign-signed and verified at boot.

C — Collector, Normalizer, Enricher

Function: receive signed telemetry securely, normalize to canonical schema, enrich with CMDB, append to lake and DB.

How: FastAPI/Gunicorn microservice with mTLS, verifies device signature against enrolled device keys, stores raw blob to S3 (encrypted), normalizes via `normalize.py`, enriches with asset metadata, inserts to ClickHouse.

Files:

- `collector/server/main.py`
- `collector/api/ingest.py`
- `collector/normalizer/normalize.py`
- `collector/enrich/enrich.py`
- `docs/API_SPEC.md` — canonical event schema

Inputs: signed event envelopes.

Outputs: raw blob in S3 & normalized event row into Event DB.

Tests: signature verification test, normalization unit tests with Windows/Linux/cloud examples, performance (events/sec).

Security: enforce mTLS, rate limiting, write-only blob storage in WORM mode optional.

D — Feature Extraction & Feature Store

Function: produce features for models (time-window aggregates, histograms, embeddings) and expose low-latency APIs for inference.

How: on-agent pre-processing for sensitive PII extraction; server-side windowing & aggregation in Spark/Beam or custom service; Redis for low-latency feature retrieval; feature store metadata.

Files:

- `collector/normalizer/features.py`
- `inference/feature_store_adapter.py`
- `infra/scripts/feature_schema.yaml`

Inputs: normalized events.

Outputs: feature vectors served via `GET /features?asset=...`.

Tests: data correctness tests, windowing boundary tests, PII scrub tests.

Security: encrypt features, RBAC for reads, store feature provenance metadata.

E — Model Families & Inference

Function: run ensemble of models (baseline autoencoder/isolation, sequence transformer, graph GNN, micro-timing detector, identity embeddings) and produce risk scores + explainability.

How: models trained in PyTorch; served via TorchServe or custom C++/Rust inference; ensemble aggregator weights models; produce SHAP or surrogate explanations.

Files:

- `inference/server/app.py`
- `inference/models/model_loader.py`
- `inference/scoring/score_request.py`
- `model-registry/service/registry.py`
- `model-registry/metadata_schema.yaml`

Inputs: feature vectors or event context.

Outputs: per-model scores + ensemble `risk_score` + `explanation` JSON.

Tests: per-model unit tests, ensemble calibration tests, latency/load tests, shadow-mode comparison.

Security: model artifacts signed, optional confidential compute (SGX/SEV) for model protection.

F — Decision Engine (Policy & Actions)

Function: apply policy DSL to risk scores; generate alerts; create action tickets with required approvals; decide low-risk auto-remediations.

How: policy engine accepts DSL rules (YAML/JSON) using thresholding, temporal windows, asset criticality; ABAC for staff. Uses `approval_token` binding for owner-required actions.

Files:

- `decision-engine/service/main.go`
- `decision-engine/policy_engine/policy.go`
- `decision-engine/playbook_runner/runner.go`
- `docs/policy_dsl.md`

Inputs: risk scores, asset context, policy configurations.

Outputs: `Alert`, `ActionTicket` objects sent to SOAR.

Tests: policy rule unit tests, approval flow tests, edge-case combinations.

Security: policies signed, policy change requires owner or admin approval depending on scope.

G — SOAR / Orchestrator (Execution & Rollback)

Function: execute playbooks (isolate host, block CIDR, revoke session), verify effects, maintain rollback artifact, interact with connectors.

How: orchestrator processes queued actions, each connector implements `apply()` and `rollback()` idempotent ops; uses per-action precheck and postcheck verification.

Files:

- `orchestrator/main.py`
- `orchestrator/connectors/*` (firewall_connector.go, cloud_connector.go, agent_connector.go)
- `decision-engine/playbook_runner/runner.go`

Inputs: `ActionTicket` with playbook steps and approval state.

Outputs: action execution logs, rollback artifacts, ledger entries.

Tests: connector mocks, transactional behavior tests, failure injection to validate rollback.

Security: connectors authenticate with short-lived tokens; owner approval required for destructive actions.

H — Guardian Core (Immutable Failsafe)

Function: minimal signed runtime that validates agent & kernel integrity, can enforce emergency lockdown, and can rehydrate system.

How: Guardian Core is a small C binary/agent with minimal API, runs with high privilege, periodically verifies agent binary signatures/hashes and kernel measurements; has locked RPCs accessible only with owner multi-sig for upgrades.

Files:

- `guardian-core/core.c`
- `guardian-core/core_service.py`

Inputs: agent heartbeat, binary checksums, kernel attestation.

Outputs: isolation commands, forensic snapshot triggers, ledger entries.

Tests: tamper simulation (modify agent) and verify Guardian Core triggers.

Security: signed binary; updates only with owner + CI cosign signatures.

I — DM-TestSecure Cyber-Range (Safe Red-Team & Synthetic Data)

Function: generate labeled synthetic telemetry mapped to MITRE ATT&CK; produce safe adversarial traces for training and validation.

How: Docker/k8s-based cyber-range with scenario YAML; `synth_generator.py` produces labeled events; DM-TestSecure uses constrained behavior (no exploitation of 3rd-party systems).

Files:

- `testsecure/cyberrange/docker-compose.yml`
- `testsecure/scenarios/*.yaml`
- `testsecure/synth_generator.py`

Inputs: recorded baseline telemetry or scenario config.

Outputs: labeled telemetry files and training datasets.

Tests: coverage tests ensuring scenarios map to MITRE techniques; label correctness checks.

Security: runs air-gapped by default in enterprise; production tests require written owner authorization.

J — Invented-Defense Pipeline (Idea → Simulate → Approve → Canary)

Function: allow the platform to propose defensive policies, simulate them, run formal safety checks, present to owner, and deploy to canary scope.

How: defense idea generator (constrained LLM + symbolic engine), simulation executor (cyber-range runner), safety invariants checker, UI review & owner sign-off, canary rollouts.

Files:

- `decision-engine/defense_pipeline/idea_processor.go`
- `testsecure/defense_simulator.py`
- `ui/web/src/pages/DefenseIdeaReview.tsx`

Inputs: defense idea cards (title, description, policy script).

Outputs: simulation report, risk/impact score, canary token for promotion.

Tests: safety invariant tests, conflict detection tests, rollback validation.

Security: idea generator limited to producing defensive constructs (no offensive code), human-in-the-loop mandatory for promotion.

K — Audit Ledger & Forensics

Function: immutably record critical events (approvals, model promotions, actions executed), provide chain-of-custody for forensics and regulators.

How: append-only ledger entries `{prev_hash, timestamp, payload, signer, signature}` stored to WORM-enabled S3 and backed up; periodic external anchors published to a public ledger or escrow service.

Files:

- `audit/ledger_service.py`
- `docs/OP_RUNBOOK.md` — ledger archiving & anchor processes

Inputs: action events, approvals, model promotions.

Outputs: ledger entries, verification utilities.

Tests: ledger integrity verification, recovery from partial corruption.

Security: ledger write API highly restricted; ledger snapshots stored offline.

L — Model Registry & Governance

Function: store model artifacts and metadata; manage promotion workflow with owner tokens; enable shadow & canary deployments.

How: model bundles stored in S3; metadata in DB; registry exposes endpoints for `POST /models/register`, `POST /models/{id}/promote` (owner token required), `GET /models/{id}/metrics`.

Files:

- `model-registry/service/registry.py`
- `model-registry/metadata_schema.yaml`
- `ui/web/src/pages/ModelReview.tsx`

Inputs: model bundle (model.bin + metadata.yaml + signature)

Outputs: model deployment to inference cluster (after signature check & canary policy).

Tests: model promotion path tests, signature verification test, canary monitoring.

Security: cosign signing mandatory; model serving nodes verify signature before loading.

M — Federated Learning & Secure Aggregation

Function: enable cross-tenant model improvements without raw data sharing using secure aggregation and differential privacy.

How: local nodes compute model deltas; deltas encrypted and aggregated via MPC/secure aggregator; central controller applies DP clipping and composes global model; owner approval required to apply.

Files:

- `model-registry/federated_controller.py`
- `docs/FEDERATED.md`

Inputs: local encrypted deltas

Outputs: aggregated model update bundle

Tests: DP guarantee tests, poisoning scenarios, aggregator correctness.

Security: opt-in only, strict provenance, owner oversight.

N — Deception & Honeypot Fabric

Function: auto-spawn honeypots tuned to attackers, capture telemetry for training and forensics, protect real assets.

How: containerized honeypots mimic services; orchestrator manages lifecycle; captured telemetry labeled and sanitized for training.

Files:

- `orchestrator/deception_manager.py`
- `testsecure/deception_scenarios/`

Inputs: suspicion triggers or proactive creation rules.

Outputs: deception telemetry, alerts.

Tests: isolation verification, capture sanitization.

Security: ensure no real secret exposure; honeypots fully isolated network-wise.

O — DLP, Insider Threat & JIT Privileged Access

Function: detect exfiltration, identify risky insiders, provide JIT privileged tokens for admin tasks.

How: monitor file operations & network egress, integrate HR signals (webhooks), issue ephemeral admin tokens via Vault on JIT approval.

Files:

- `collector/normalizer/dlp.py`
- `decision-engine/insider_detector.py`
- `orchestrator/jit_access.py`

Inputs: file access events, network egress events, HR signals

Outputs: insider risk alerts, JIT tokens

Tests: exfiltration simulation, HR derived event processing

Security: content hashing and redaction; legal hold.

P — Compliance & Reporting Automation

Function: map detections and evidence to compliance controls (NIST/ISO/PCI/HIPAA), produce redacted evidence bundles.

How: metadata mappings; automated evidence collection with chain-of-custody; compliance dashboards.

Files:

- `docs/compliance_mappings.yaml`
- `ui/web/src/pages/ComplianceDashboard.tsx`

Inputs: alerts, ledger entries

Outputs: compliance report bundles, control status

Tests: generate sample report for simulated incident and verify control mapping.

Security: redaction & DPA workflows in place.

Q — CI/CD, SBOM, cosign & Secure Release

Function: reproducible builds, SBOM generation, artifact signing, two-step release (CI + owner signature).

How: GitHub Actions / GitLab CI: build, test, generate SBOM, cosign sign image, push to registry; release channel gating requires owner cosign.

Files:

- `ci/github-actions/build.yml`
- `ci/github-actions/sign-release.yml`
- `ci/scripts/sbom-generate.sh`
- `ci/scripts/cosign-sign.sh`

Tests: reproducible build tests, SBOM coverage, signature verification in runtime.

Security: owner key custody critical; use HSM or hardware token.

R — Offline / Air-gapped Update Path

Function: accept signed update bundles via physical media; verify signature; run sandboxed canary; promote.

How: bundle format `bundle.tar.gz` contains `metadata.yaml`, `model.bin`, `signature.sig`; offline verification tool verifies cosign signature and owner token.

Files:

- `infra/airgap/usb_bundle_spec.md`
- `docs/OFFLINE_SPEC.md`
- `collector/api/offline_ingest.py` (or UI upload flow)

Inputs: signed update bundle via USB

Outputs: applied update (after sandbox pass)

Tests: offline ingestion test, signature verification, rollback.

Security: require owner key signature; audit ledger entry for manual update.

7 — Models, training, validation, anti-poisoning & governance

Model families (recap + implementation)

- **Per-host baselines**: autoencoders/isolation forest (Python / PyTorch / scikit-learn).
- **Sequence models**: Transformers or LSTMs for time-series of events.
- **Graph models**: GNN for lateral movement detection (DGL/PyG).
- **Identity/intent**: embeddings (Siamese networks), classifier for intent.
- **Micro-timing**: statistical detectors in low-latency C/Go.
- **Rule/signature engine**: YARA-like.
- **LLM assistant**: small local models (Llama2-type or local custom) for narrative generation, limited to safe operations in sandbox before invoking.

Training & validation pipeline

- **Data collection**: signed telemetry + DM-TestSecure synthetic labeled traces + human labels.
- **Sanitization**: remove PII, anonymize identifiers, tag provenance.
- **Data split**: time-aware splits to avoid leakage.
- **Adversarial tests**: DM-TestSecure generates adversarial traces (safe) to test resilience.
- **Metrics**: precision, recall, F1, calibration, false positives per 1k assets/day, business impact simulations.
- **Explainability**: SHAP or surrogate explanations required for all models deployed.
- **Canary & shadow**: models must pass shadow tests (scoring-only) for N days and canary low-impact action tests before full deployment.
- **Approval**: owner reviews metrics & signs promotion token.

Anti-poisoning & provenance

- All training samples for cross-tenant learning require provenance (who, which agent, signed). Discard or downweight samples lacking provenance. Use robust loss functions, clipping, and DP on aggregation. Run poisoning detection (outlier detector on training set) and adversarial robustness evaluation.

8 — Self-healing, fail-safes & recovery

Self-healing tiers

- **Tier 0**: service restart.
- **Tier 1**: restore signed config from local snapshot.
- **Tier 2**: re-deploy signed container image.
- **Tier 3**: isolate node (read-only) and require owner reactivation; Guardian Core enforces.

Mechanisms

- Health telemetry + SLO monitoring triggers playbooks for healing.
- Each automated healing step creates rollback artifact (snapshot) and ledger entry.
- Emergency restore requires owner multi-sig for critical operations.

Recovery steps

- For catastrophic loss: bootable signed recovery image → owner signs rehydrate token → system replays ledger & rehydrates assets.

9 — Invented-defense pipeline & DM-TestSecure (detailed)

Idea generation

- Constrained LLM + symbolic template engine produce policy suggestions; generator limited to defensive constructs only.

Simulation

- Automatically translate idea to runnable policy script (WAF rule, rate-limit config, ephemeral token generation) and run scenario in cyber-range replaying recent sanitized telemetry.

Safety checks

- Formal invariants (no data loss), resource bounds, compliance checks (e.g., HIPAA), rollback existence, performance impact thresholds.

Owner approval & canary

- Present simulation summary & metrics for owner review; owner signs promotion token to allow canary rollout for selected assets; monitor for pre-set period; promote or rollback automatically.

10 — Audit, SBOM & supply-chain signing

SBOM & reproducible builds

- Generate SBOM via `syft` or similar; include SBOM in release artifacts and store in artifact registry.

Signing & release

- cosign for image signing; require two signatures for production release: CI + owner. Runtime verifies cosign signature before execution.

Ledger anchoring

- Weekly anchor of ledger snapshot to public immutable ledger (optional) or external escrow for strong tamper evidence.

11 — Owner trust model (enrollment, approvals, emergency)

Enrollment

- Owner enrolls device via `POST /owner/enroll/start` (returns WebAuthn challenge); device attestation verified; `owner_record` stored.

Approvals

- For major actions (model promotion, production playbook with irreversible action), create `approval_ticket` in Decision Engine; owner signs challenge via device -> `approval_token` issued -> orchestrator verifies token before executing.

Emergency

- Emergency recovery seeds stored encrypted in HSM and optionally physical USB. Emergency unlock requires multi-signature owner + auditor or owner + legal (configurable).

12 — Deployment models, infra & hardware

Minimum prototype hardware

- Laptop (16–32GB RAM), Redmi 10C for WebAuthn PoC, external disk for logs.

Pilot infra (on-prem/hybrid)

- k8s cluster with namespace separation (inference, training, api, registry).
- Kafka for ingest (or managed equivalent), ClickHouse for events, S3 for blobs, Redis for features.
- GPU nodes for training (NVIDIA A100/RTX 6000).

Enterprise appliances

- 2U rack server appliance with TPM2.0, HSM connectivity, NVMe, 10Gbps uplink, pre-signed OS images and Guardian Core pre-flashed.

Confidential compute

- Optional SGX/SEV or cloud confidential VMs for model decryption in trusted enclave.

13 — APIs, schema & playbook catalog

Core APIs (representative list)

- `POST /owner/enroll/start`
- `POST /owner/enroll/complete`
- `POST /telemetry/event`
- `POST /telemetry/batch`
- `POST /score`

- `POST /models/register`
- `POST /models/{id}/promote` (owner token)
- `POST /playbooks/execute`
- `POST /actions/{ticket}/rollback`
- `GET /alerts`
- `GET /audit/ledger`

Canonical event schema (see Appendix)

- `event_id`, `timestamp_utc`, `asset_id`, `event_type`, `payload`, `payload_hash`, `signature`, `provenance`, `privacy_tags`

Playbook example: `isolate_endpoint`

(Full playbook JSON provided in appendix; includes required approvals and rollback steps.)

14 — CI/CD, SBOM, cosign & release governance

Pipeline

1. Commit → CI builds container images & artifacts → run unit/integration tests → generate SBOM → run cosign to create CI signature.
2. Publish artifact in `staging` channel → owner reviews model/changes in UI → owner cosigns (via secure hardware token) → artifact promoted to `release`.
3. Runtime verifies cosign signatures before load.

Files

- `ci/github-actions/build.yml`
- `ci/scripts/sbom-generate.sh`
- `ci/scripts/cosign-sign.sh`

15 — Security threat matrix (condensed)

We already listed 20 threats earlier. DM-Protect implements defenses covering each: model poisoning, supply-chain, insider, evasion, rootkits, agent tampering, cloud partition, credential theft, lateral movement, zero-day, fileless malware, ransomware, configuration drift, TLS interception, log forging, rogue admin, model extraction, firmware compromise, dependency compromise, social engineering.

The platform's defenses are: multi-model detection, Guardian Core, signed supply chain, owner control, offline mode, deception, anti-poisoning, audited ledger, and self-healing.

16 — Compliance, privacy & legal

- **Contracts**: DPA, pen-test authorization template, EULA with owner rights & escrow policy.
- **Certifications**: roadmap to SOC2 → ISO27001 → FedRAMP as needed.
- **Privacy**: local-first processing, PII redaction, opt-in federated learning with DP, legal hold.
- **Regulated data handling**: HIPAA/PCI mappings & playbooks for breach reporting.

17 — Observability, KPIs & dashboards

Key metrics

- Model metrics: precision, recall, F1, calibration, false positive rate per 1k assets/day.
- Operational: ingestion lag, agent heartbeat %, inference latency, canary success rate.
- Business: MTTD, MTTR, incidents prevented, analyst time saved.
- Safety: % of auto-actions reverted, rollback frequency, owner approvals rate.

Dashboards

- CISO dashboard: business risk, trending threats, compliance posture.
- SOC workbench: alerts, case timelines, action history.
- Owner console: pending approvals, last 24h major actions, ledger snapshots.

18 — Operations, runbooks & staffing

Roles

- Product owner (you) — root owner privileges.
- Security architect — design & audits.
- ML engineers — model dev & governance.
- Backend engineers — collector, inference, orchestrator.
- Agent engineers — cross-platform telemetry.
- DevOps / SRE — infra & CD pipelines.
- SOC analysts / incident responders — operate workbench.
- Legal & compliance — DPA, audits.
- Sales & customer success — enterprise onboarding & certification.

Runbooks (examples)

- Onboarding: install agent monitor-only, collect 7–30 days baseline, run DM-TestSecure profile, owner sign baseline.
- Incident: triage in workbench, decide manual vs auto remediation, execute playbook, record to ledger, root-cause & retrospective.
- Model promotion: sandbox → owner review → owner sign → canary monitoring → full promote.

19 — Developer file map (complete repo layout & purposes)

Full repo layout (same as earlier but expanded). Each file and folder is **required** for the full system. Implement in the order below for speed. (I can generate scaffolding files next.)

```
dm-protect/
├─ infra/
│  ├─ k8s/
│  ├─ terraform/
│  └─ airgap/
├─ agent/
│  ├─ cmd/agent/main.go
│  ├─
pkg/collection/{process_collector.go,netflow_collector.go,syscall_coll
ector.go}
│  ├─ pkg/signer/device_signer.go
│  ├─ pkg/enforce/isolate.go
│  ├─ pkg/health/heartbeat.go
│  └─ config/agent.yaml
├─ mobile-agent/
│  ├─ android/(Kotlin app)
│  └─ ios/(Swift app)
├─ collector/
│  ├─ server/main.py
│  ├─ api/{ingest.py,enroll.py,offline_ingest.py}
│  ├─ normalizer/{normalize.py,features.py,dlp.py}
│  └─ enrich/enrich.py
├─ inference/
│  ├─ server/app.py
│  ├─ models/model_loader.py
│  ├─ scoring/score_request.py
│  └─ feature_store_adapter.py
├─ decision-engine/
│  ├─ service/main.go
│  ├─ policy_engine/policy.go
│  ├─ playbook_runner/runner.go
```

```
│  └─ approval/approval.go
├─ orchestrator/
│  ├─ main.py
│  ├─
connectors/{firewall_connector.go,cloud_connector.go,agent_connector.g
o}
│  └─ deception_manager.py
├─ model-registry/
│  ├─ service/registry.py
│  ├─ federated_controller.py
│  └─ metadata_schema.yaml
├─ testsecure/
│  ├─ cyberrange/docker-compose.yml
│  ├─ scenarios/*.yaml
│  └─ synth_generator.py
├─ ui/
│  ├─ web/package.json
│  ├─ web/src/App.tsx
│  └─
web/src/pages/{Dashboard,OwnerApproval,ModelReview,DefenseIdeaReview,C
omplianceDashboard}.tsx
├─ guardian-core/
│  ├─ core.c
│  └─ core_service.py
├─ audit/
│  └─ ledger_service.py
├─ ci/
│  ├─ github-actions/{build.yml,sign-release.yml}
│  └─ scripts/{sbom-generate.sh,cosign-sign.sh}
├─ docs/
│  ├─ ARCHITECTURE.md
│  ├─ OFFLINE_SPEC.md
│  ├─ OWNER_ONBOARDING.md
│  ├─ OP_RUNBOOK.md
│  ├─ API_SPEC.md
│  ├─ PKI.md
│  └─ FEDERATED.md
└─ README.md
```

Each file's purpose is described earlier in each feature section.

20 — Exact prioritized build roadmap & day-by-day initial plan

Phase 0 — Day 0–10 (PoC)

- Create repo skeleton and CI skeleton.
- Implement minimal `collector/server/main.py` with `POST /telemetry/event` (HMAC signing for PoC).
- Implement `agent/main.go` proto that collects `ps` and POSTs signed events.
- Implement `inference/app.py` with toy isolation forest trained on synthetic data.
- Implement `decision-engine` stub that prints alerts.
- Build simple React dashboard to show alerts.
- Run a local docker-compose for these services.

Phase 1 — Week 2–8 (MVP)

- Harden signature verification: implement device key management & WebAuthn enroll endpoints.
- Add ClickHouse for events and S3 for blobs.
- Add basic model registry & cosign signing stub.
- Implement Guardian Core minimal check.
- Build DM-TestSecure basic cyber-range and synthetic generator.

Phase 2 — Month 2–6 (Scale & Security)

- Add full inference cluster, multi-model families, SHAP explainability.
- Implement SOAR connectors, playbook editor, and orchestrator rollback.
- Integrate HashiCorp Vault & HSM for key storage.
- Implement cosign signed CI/CD and SBOM generation.

Phase 3 — Month 6–12 (Enterprise)

- Confidential compute support, federated learning prototype, compliance modules (SOC2, ISO), voice assistant and physical sensor integration, deception fabric.

21 — Tests & verification suite

- Unit tests for every component (90% target).
- Integration tests covering Agent → Collector → Inference → Decision → Orchestrator.
- Shadow model tests: new models run in scoring-only mode for N days to collect delta metrics.
- Adversarial tests: DM-TestSecure produces adversarial traces; validate model resilience.
- Chaos tests: simulate network partitions and service crashes; validate self-heal.
- Ledger tests: append/verify and anchor tests.
- Supply chain tests: reproducible build, SBOM completeness, verify cosign on runtime.

22 — Appendix (sample artifacts & commands)

Sample event envelope

(see earlier in conversation — include canonical JSON with `payload_hash` and `signature`).

Sample playbook: `isolate_endpoint`

(Full JSON included in prior messages; ensure required approvals and rollback actions).

Starter commands (create repo skeleton)

```
mkdir dm-protect && cd dm-protect
git init
# create directories
mkdir -p infra agent collector inference decision-engine orchestrator
model-registry testsecure ui guardian-core audit ci docs
# create placeholder README and docs
touch README.md docs/ARCHITECTURE.md docs/OFFLINE_SPEC.md
docs/OWNER_ONBOARDING.md
```

PoC run (after starter code)

1. Start collector: `python3 collector/server/main.py`
2. Start inference: `python3 inference/server/app.py`
3. Start decision engine: `go run decision-engine/service/main.go`
4. Start agent: `go run agent/cmd/agent/main.go`
5. Open UI at http://localhost:3000 and observe alerts.

Final notes, caveats & next steps

- This document is exhaustive and intended as the full engineering and security blueprint. It is **already enterprise-grade**, but building and operating DM-Protect at bank/military scale requires a strong team, third-party audits, and legal processes (pen-test authorization, DPA, etc.).
- I can now:
    - Generate the **full repo scaffold** (empty files and readmes) for you to clone.
    - Generate the **PoC starter code** for the minimal end-to-end pipeline (collector, agent, inference, decision, UI) and exact run instructions.
    - Generate a **formal PDF Master Blueprint** suitable for sharing with auditors or investors.
    - Produce the **first priority code files** with implementation (e.g., `collector/main.py`, `agent/main.go`, `inference/app.py`, `decision-engine/main.go`, UI Dashboard).

Tell me **which artifact you want me to produce now** (I recommend “Generate PoC starter code” + “Repo scaffold”), and I’ll output runnable code and commands immediately.