System Requirements Document

Autonomous API Red Team (AART)

Hybrid Graph + Symbolic-Lite Core | Universal Repo Coverage | Agentic Flow

v2.0 — Universal Coverage Edition

---

1. Product Requirements

1.1 Purpose

Build a developer-first SaaS (AART) that continuously reasons about API-first web apps, generating deterministic exploit hypotheses via a graph + symbolic-lite engine. AART is universally scoped — it delivers meaningful security value on repos of any size, from a single-file Express app to a complex multi-tenant API. High-confidence hypotheses are selectively validated in isolated sandboxes, with reproducible proof-of-exploit, minimal patch suggestions, and longitudinal threat memory.

1.2 Goal / Success Criteria

- Deliver deterministic, reproducible exploit proofs for access-control issues (IDOR, horizontal/vertical privilege escalation, mass-assignment) on any repo regardless of size.
- Produce a first meaningful finding within 3 minutes for simple repos and 8 minutes for medium repos.
- Keep false-positive confirmed exploits ≤ 5% after runtime validation.
- Provide plain-English PR feedback with reproducible steps, suggested patch diff, and confidence score accessible to developers of all experience levels.
- Maintain strong safety: no tests against production, strict sandboxing, no secrets exfiltration.

1.3 Scope

MVP (Must-Have)

- Repo ingestion (GitHub) and AST-based route extraction for Node.js/Express.
- Complexity Router: auto-detects repo tier (simple / medium / complex) and routes to appropriate pipeline.
- Heuristic Fast-Path Scanner: lightweight static checks for simple repos. No graph or sandbox required. Runs in < 3 minutes and always produces at least advisory-level findings.

- Attack Surface Graph builder (routes, middleware, models, permissions) for medium/complex repos.
- Symbolic-lite engine for ownership/role constraint modeling.
- Attack templates: IDOR, horizontal escalation, simple mass-assignment, role bypass patterns.
- Sandbox runner (Docker) to validate high-confidence candidates for medium/complex repos.
- PR integration (GitHub App) posting plain-English structured comments.
- Threat Memory DB to store patterns per-app fingerprint and complexity tier.

Phase 2 (After MVP)

- Auto-PR patch drafts, more languages/frameworks (FastAPI, Django, Fastify), GraphQL support.
- Expanded attack templates, persistent agent learning, enterprise features (SSO, RBAC).
- ATT&CK mapping & compliance reporting.
- Security health grade dashboard and trend tracking.

1.4 Key Functional Requirements

1. Complexity Detection & Routing: On ingestion, classify repo as simple (≤10 routes), medium (10–50 routes), or complex (50+ routes, multi-role). Route to fast-path or full pipeline accordingly.
2. Heuristic Fast-Path Scanner: For simple repos, run static pattern checks (missing auth middleware, unguarded ID params, obvious mass-assignment) without graph or sandbox. Always produce at least one advisory-level finding or a "no critical issues — here's what we checked" summary.
3. Ingest repository and produce AppGraph: Parse routes, middleware, controllers, model access. Detect auth middleware and likely ownership checks. Required for medium/complex repos; optional enrichment for simple repos.
4. Construct attack goals and candidate paths: Goals: read-other-user-data, escalate-role, access-admin-only-endpoint, mass-assignment. Paths generated deterministically from graph traversal + symbolic constraints.
5. Symbolic-lite constraint evaluation: Evaluate reachability and variable comparisons. Produce deterministic feasibility score (0..1).
6. LLM-constrained semantic interpreter: Called only to interpret ambiguous code blocks, returning structured JSON hint. Deterministic engine must validate or reject.
7. Selective runtime sandbox validation: If deterministic + symbolic confidence ≥ T2 (e.g., 0.75), spawn sandbox, seed DB, run exploit attempt, capture evidence.
8. Plain-language reporting: Every finding must include a one-sentence plain-English summary of the impact (e.g., "Anyone can read any user's invoice"), not just technical identifiers.
9. Patch suggestion and verification: LLM drafts minimal code fix per constrained schema; deterministic engine or CI tests verify fix in sandbox.

10. Human-in-loop gating: Developer must accept any auto-PR before merge. No automatic production changes.
11. Threat Memory & prioritization: Store exploit templates that succeeded; bias future exploration by tier.
12. Integration: GitHub App, CLI, web UI dashboard, Slack/Jira connector.

1.5 Non-Functional Requirements

- Safety & Isolation: Sandboxes have no external network egress and run with strict resource limits.
- Universal Performance: Simple repos ≤ 3 min; medium repos ≤ 8 min; complex repos ≤ 15 min.
- No repo left behind: Every ingested repo receives at minimum a heuristic scan result.
- Scalability: Horizontal scaling of sandbox workers and graph processors.
- Auditability: Full audit logs of all decisions, LLM calls, exploit evidence.
- Explainability: Every confirmed exploit includes deterministic trace + plain-language summary.
- Privacy: No production secrets used; redaction when user-provided data exists.

1.6 Personas & User Stories

- Dev (bootcamp grad): On PR, gets a plain-English comment "Anyone can read any user's profile" with a one-line fix. No security jargon.
- Maya (solo founder): Immediate feedback on her 3-route Express app within minutes of connecting her repo.
- Ravi (engineering lead): Weekly digest of confirmed exploits and prioritized remediation backlog for a 60-route API.
- Ammar (agency dev): CLI-based pre-client-demo scan producing evidence to justify fix time, works on both tiny and complex repos.

1.7 Acceptance Criteria

- A minimal Node.js/Express app (1 route, 1 model) produces at least one advisory or confirmed finding within 3 minutes.
- For 10 representative repos (mix of simple, medium, complex): every repo produces at least one result.
- Sandbox validation confirms ≥ 80% of high-confidence candidates.
- All PR comments include plain-English summary, reproducible steps, diff suggestion, and confidence score.
- All LLM outputs are schema-validated and logged.

2. Workflow Analysis

2.1 Onboarding Flow

13. Developer installs GitHub App & grants repo access.
14. AART ingestion worker clones repo and runs Complexity Router.
15. Simple repos: fast-path heuristic scanner runs immediately. Result delivered within 3 minutes.
16. Medium/complex repos: full AST parse, graph build, and symbolic evaluation run.
17. System returns App Fingerprint: stack, auth type, detected models, complexity tier, security health grade.
18. Developer sees initial dashboard with findings and threat memory summary.

Key UX: Plain-English finding descriptions, security health grade, ability to mark false positives, per-repo config.

2.2 PR / CI Flow (Happy Path)

19. Developer opens PR. GitHub App triggers AART check.
20. Complexity Router determines which pipeline to run (delta only for changed routes).
21. Simple tier: heuristic scanner checks changed routes within 2 minutes.
22. Medium/complex tier: Deterministic engine builds AppGraph delta; symbolic engine evaluates candidates.
23. High-confidence candidates (score $\geq$ T2) trigger sandbox runner.
24. If exploit confirmed: PR comment with plain-English summary, evidence, suggested patch, `aart/confirmed` label, optional blocking. If borderline: low-confidence advisory (non-blocking, clearly labeled).
25. Developer applies patch -> AART re-runs; if fixed, marks resolved and updates threat memory.

2.3 Periodic Scans & Continuous Monitoring

- Nightly/weekly scheduled scans for all repo tiers (opt-in).
- Threat Memory influences scan priority (recurring weaknesses prioritized).
- Dashboard shows security health grade trend over time.
- Centralized view of confirmed exploits & remediation tasks; export to Jira.

2.4 Incident Simulation / Response

- If a confirmed exploit exists, generate an Incident Report (executive + technical).
- Provide timeline, impacted assets, proof of exploit, and recommended mitigations.
- "Drill" mode: simulate exploit without producing external notifications.

2.5 Human-in-the-Loop Controls

- Approve auto-PRs before merge.
- Mark finding as ignored (with reason) or false positive (for learning).
- Set sandbox compute budgets per repo/org.
- Override complexity tier if auto-detection is wrong.

---

3. Architectural Design

3.1 High-Level Components

Component	Responsibility
Ingestion Worker	Clones repo, normalizes code, triggers AST extraction and complexity detection
Complexity Router	Classifies repo as simple/medium/complex; routes to appropriate pipeline
Heuristic Fast-Path Scanner	Lightweight static checks for simple repos: missing auth, unguarded IDs, mass-assignment patterns. No graph or sandbox.
AST & Parser Service	Babel/SWC (JS/TS); produces normalized AST for graph builder
Graph Builder	Constructs AppGraph (routes, middleware, models, auth nodes) for medium/complex repos
Symbolic-Lite Engine	Lightweight constraint modeler; evaluates ownership checks and reachability
LLM Reasoner (Constrained)	Schema-wrapped interpreter + patch suggester. Stateless; output validated by deterministic engine.
Attack Planner	Ordered candidate exploit plans from templates + graph
Sandbox Orchestrator	Docker engine pool, sandbox templates, test data seeders. Used for medium/complex only by default.
Execution Engine	Runs requests, verifies responses, collects logs & artifacts
Threat Memory DB	Stores app fingerprints, exploit patterns, success rates, complexity tier
Policy & Governance	Confidence thresholds, cost caps, RBAC, audit logging
Integrations	GitHub App, CLI, Slack/Jira connectors
Frontend Dashboard & API	Show findings, health grade, evidence, allow approvals

3.2 Data Flow (Sequence)

26. GitHub webhook triggers Ingestion Worker; repo is cloned.
27. Complexity Router classifies repo (simple / medium / complex).

28. Simple path: Heuristic Fast-Path Scanner runs static checks. Produces advisory findings directly. Skip to step 8.
29. Medium/complex path: AST Service parses code; Graph Builder stores AppGraph.
30. Symbolic Engine evaluates constraints; candidate list created with feasibility scores.
31. For candidates with score ≥ T1 (e.g., 0.5): LLM Reasoner queried for structured semantic hint.
32. Deterministic validator merges LLM hint into final confidence score.
33. If score ≥ T2 (e.g., 0.75): Sandbox Orchestrator spins container; Execution Engine runs scenario.
34. Results (PASS/FAIL + artifacts) stored in Threat Memory & presented in UI.
35. GitHub App posts PR comment (plain-English summary + technical evidence).
36. Developer applies fix; AART re-runs and marks resolved.

Thresholds: T1 = 0.5 (LLM assist trigger), T2 = 0.75 (sandbox trigger). Both configurable per repo.

3.3 Tech Stack

- Backend: Python (FastAPI) or TypeScript (NestJS)
- AST: Babel / SWC (JS/TS), Esprima for JS AST
- Graph DB: Neo4j or in-memory store with Postgres persistence
- Symbolic Engine: Custom lightweight evaluator (Python); no heavy Z3 initially
- Sandbox: Docker with rootless containers, ephemeral networks, test seeding harness
- LLM: Provider-agnostic wrapper with strict JSON-schema validation
- Storage: PostgreSQL (metadata), S3 for artifacts, Redis for queues (Celery/RQ)
- Observability: Prometheus + Grafana, structured logs (ELK)
- CI/Integration: GitHub App, GitHub Actions templates, Slack/Jira connectors

3.4 Deployment & Scaling

- Stateless services autoscale behind API gateway.
- Sandbox worker pool scales horizontally (queue depth + cost budget).
- Fast-path scanner is CPU-only, cheap to scale — optimized for the mass market.
- AppGraph snapshots persisted; Threat Memory partitioned by org and tier.
- Multi-tenant per-org quotas; VPC/On-prem options for enterprise.

---

4. Agentic Flow — Detailed

4.1 Agent Roles

- Complexity Classifier Agent: detects repo tier and routes to appropriate pipeline on every ingestion.
- Heuristic Scanner Agent: for simple repos, runs static pattern checks and generates advisory findings without graph or sandbox.
- Recon Agent: builds AppGraph, finds entry points and auth middleware (medium/complex repos).
- Planner Agent: enumerates candidate exploit paths using templates + symbolic reasoning.
- Reasoning Agent (LLM): resolves ambiguous checks and drafts fix diffs (only when called, schema-constrained).
- Executor Agent: runs sandbox-based tests and verifies outcomes.
- Remediation Agent: drafts PR with patch (developer approval required).
- Memory Agent: updates Threat Memory and biases future plans by tier and fingerprint.

4.2 Simple Repo Fast-Path Example

Scenario: A 3-route Express app with no auth on one endpoint.

37. Complexity Classifier: 3 routes detected -> simple tier -> route to Heuristic Scanner.
38. Heuristic Scanner: static AST check detects GET /users/:id has no auth middleware applied.
39. Advisory finding generated: "GET /users/:id is publicly accessible with no authentication. Anyone on the internet can read user data." Confidence: deterministic 0.95 (static, no ambiguity).
40. No sandbox needed. PR comment posted within 2 minutes.
41. Suggested fix: apply authMiddleware to route (LLM-drafted one-liner diff).
42. Developer applies fix -> AART re-runs heuristic check -> passes -> marked resolved.

4.3 Full Pipeline Example: IDOR Detection (Medium/Complex Repo)

43. Recon: Parse repo -> identify GET /invoices/:id reads Invoice model; no ownership check detected. AppGraph node created.
44. Symbolic: req.user.id == invoice.userId branch not present in route handler AST. Feasibility = 0.68.
45. LLM Reasoning: called with code snippet; returns structured hint:

```
{ "ownership_inference": false, "suggested_check": "if (invoice.userId !==
req.user.id) throw 403", "confidence": 0.85, "explain": "No direct equality comparison
found." }
```

46. Engine merges LLM hint -> final confidence 0.74. Additional heuristic check raises to 0.78 -> schedule sandbox.
47. Executor: spin container, seed DB with users A and B, create invoice owned by B. Auth as A, request /invoices/{B_invoice_id}. Response contains B's data -> PASS.

48. Report: plain-English summary + deterministic trace + request/response evidence + LLM-drafted patch diff. Confidence 0.92.
49. Developer applies patch -> re-run -> FAIL (exploit prevented). Threat Memory updated.

4.4 LLM Interaction Schema (Constrained)

All LLM responses must match a strict JSON schema. Validator rejects anything not matching schema or with suspicious content (e.g., shell commands).

Interpretation schema:

```
{ "ownership_check": boolean, "ownership_field": string|null, "confidence": number,
"explanation": string }
```

Patch suggestion schema:

```
{ "patch_diff": string, "tests_added": string|null, "confidence": number }
```

4.5 Failure Modes & Mitigations

- Hallucinated exploit (LLM): Deterministic engine verifies; runtime validation required. LLM overconfidence ignored unless deterministic checks pass.
- Sandbox escape risk: Rootless containers, Seccomp, AppArmor, no network egress, non-root, resource caps.
- High cost from wide exploration: Budget quotas, priority by threat memory, confidence-first strategy. Fast-path avoids sandbox for simple repos.
- Simple repos produce no results: Heuristic layer always returns at least advisory-level findings or a "what we checked" summary. No silent passes.
- False-negatives (missed exploit): Expand templates iteratively; manual scan triggers; user feedback feeds improvement.
- Developer distrust due to noise: Strict confidence thresholds, clear evidence, easy FP marking.

5. Data Models & API

5.1 AppGraph Node

```
{ "id": "route:/invoices/:id", "type": "endpoint", "method": "GET", "auth_required":
true, "roles": ["user"], "accesses": ["model:Invoice"], "ownershipField":
"invoice.userId", "source_files": ["controllers/invoiceController.js:45"] }
```

5.2 ExploitCandidate

```
{ "id": "exploit_0001", "appFingerprint": "sha256:abc", "repoTier": "medium", "type":
"IDOR", "route": "route:/invoices/:id", "symbolicScore": 0.68, "llmHintConfidence":
0.85, "finalConfidence": 0.74, "status": "queued" }
```

5.3 ConfirmedExploit

```
{ "id": "exp-confirm-0001", "repoTier": "medium", "plainLanguageSummary": "Anyone can
read anyone else's invoice.", "evidence": { "request": "curl -X GET ...", "response":
"{ invoice: {...}}", "logs": "..." }, "trace": [{"from": "route:/invoices/:id", "to":
"model:Invoice", "note": "no ownership check"}], "suggestedPatchLink": "https://...",
"createdAt": "2026-02-22T..." }
```

5.4 HeuristicFinding (Simple Repos)

```
{ "id": "heuristic-0001", "repoTier": "simple", "type": "missing-auth", "route": "GET
/users/:id", "plainLanguageSummary": "GET /users/:id is publicly accessible — anyone
can read user data.", "confidence": 0.95, "suggestedFix": "Apply authMiddleware to
this route.", "status": "open" }
```

5.5 GitHub PR Comment Payload

- Plain-English title & one-line impact summary.
- Severity, reproducible steps (sanitized curl), artifact links.
- Suggested patch diff, confidence score, `aart/confirmed` label recommendation.
- Buttons: Create Fix PR, Mark false positive, Re-run check.

---

6. UX & UI / PR Workflows

PR Comment Templates

Simple Repo (Heuristic Finding)

Title: Missing Authentication — GET /users/:id

"GET /users/:id has no authentication. Anyone on the internet can read any user's data without logging in."

Confidence: Deterministic 0.95 (static analysis)

Suggested fix: Apply your auth middleware to this route (one-line diff attached).

Medium/Complex Repo (Confirmed IDOR)

Title: Confirmed IDOR — GET /invoices/:id (runtime validated)

"User A could access User B's invoice — proof included."

Repro steps: sanitized curl + expected response. Evidence: response snippet, logs.

Suggested minimal patch (link to auto-PR draft). Confidence: 0.92 | runtime PASS.

Buttons: Create Fix PR | Mark false positive | Re-run check

Dashboard Views

- Overview: security health grade, total confirmed exploits, trends, open remediation items, breakdown by repo tier.
- Repo page: app fingerprint, complexity tier, recent findings, threat memory summary, health grade trend.
- Exploit detail: deterministic trace graph, symbolic constraints, runtime logs, patch diff editor, plain-language impact statement.

CLI

```
aart scan --repo . --mode local          # auto-detects tier, runs appropriate pipeline
aart scan --repo . --tier simple         # force fast-path heuristic scan only
aart scan --repo . --tier full           # force full graph + sandbox pipeline
```

---

7. Testing, QA & Validation Plan

Unit / Integration Tests

- Heuristic scanner tests across minimal single-file Express apps.
- AST parser unit tests across representative frameworks & coding styles.
- Symbolic engine tests for edge-condition logic.
- LLM schema validator tests with adversarial inputs.
- Sandbox harness tests for no-evidence leakage and repeatability.
- Complexity Router tests verifying correct tier classification across a diverse repo set.

Acceptance Tests

- Curated set of synthetic vulnerable apps: 5 simple, 3 medium, 2 complex (OWASP Broken Access Control patterns).
- Every simple app must produce at least one advisory or confirmed finding.
- Measure confirmation rates and false-positive rates across all tiers.

Security Tests

- Internal red-team to attempt sandbox escape.

- Fuzzing the LLM input & validator to prevent injection.

---

8. Governance, Audit & Compliance

- Audit logs: immutable logs for every decision across all tiers (heuristic result, graph snapshot, symbolic result, LLM response, sandbox output).
- Privacy: allow orgs to opt for on-prem components for sensitive code.
- Data retention: configurable (30/90/365d).
- RBAC: org-admin, security-reviewer, developer roles.
- Regulatory: enterprise VPC/On-prem option for SOC2/GDPR-sensitive customers.

---

9. KPIs, Metrics & Observability

Product KPIs

- Repos onboarded by tier (simple / medium / complex)
- First-finding time by tier (target: < 3 min simple, < 8 min medium)
- Heuristic advisory findings per simple repo (must be > 0 for all repos)
- Confirmed exploits / active repo / month
- False positive rate (confirmed exploit labeled FP by developer)
- Mean time from exploit confirmation to fix
- Free-to-paid conversion rate

Operational

- Sandbox cost per confirmed exploit
- Heuristic scanner cost per repo (should be near-zero for simple tier)
- Queue length & average processing latency by tier
- Number of auto-PRs created and accepted
- LLM call count and schema reject rate

---

10. Build Priorities (Roadmap)

50. Complexity Router + Heuristic Fast-Path Scanner (delivers value on simple repos from day one).
51. AST parser, AppGraph, and symbolic-lite engine for Node.js/Express.

52. Attack templates + attack planner.
53. Sandbox runner and test harness.
54. GitHub App for PR checks and PR comments (plain-language template).
55. Constrained LLM interface for interpretation and patch drafting.
56. Threat Memory DB & prioritization logic.
57. Dashboard (health grade view) + CLI + initial beta customers.

11. Risks & Mitigation Summary

- Safety risk (sandbox escapes): rootless containers, no external egress, strict seccomp/AppArmor.
- Hallucination risk (LLM): strict schema validation; deterministic engine override; runtime validation required.
- Cost risk (too many sandboxes): budget quotas, prioritize via Threat Memory, fast-path avoids sandbox for simple repos.
- Universal coverage risk (simple repos silently skipped): Heuristic layer is a first-class pipeline — monitored, tested, and measured separately.
- Developer trust: clear evidence, low-FP rate, plain-language explanations, easy feedback loop.
- Coverage risk (missed exploit families): iterative template expansion, customer-driven rule addition.