Product Requirements Document

Autonomous API Red Team (AART)

Continuous Autonomous Red Teaming for API-first Web Apps — Universal Coverage from Simple to Complex Repos

Executive Summary

AART is a SaaS that continuously simulates attacker behavior against API-first web applications of any size and complexity — from a solo developer's weekend project to a multi-service startup backend. AART finds real exploit paths (IDOR, broken access control, privilege escalation, mass-assignment), verifies them safely in ephemeral sandboxes, and proposes minimal, testable fixes.

The product is built as a hybrid system: a deterministic graph + symbolic-execution core, LLM-assisted semantic reasoning, and selective runtime validation for high-confidence candidates. Crucially, AART is designed to be universally accessible — delivering instant value on even the simplest two-route Express app while scaling gracefully to complex multi-role, multi-tenant APIs.

Key positioning: "Your autonomous, safe red-team teammate — for every developer, every repo, every PR."

Objectives & Success Metrics

Primary Objectives

- Deliver real security value on repos of any complexity — from a single-file Express server to a large multi-service API.
- Reduce time-to-fix for high-impact access-control issues across the full spectrum of customer codebases.
- Provide reproducible exploit proofs that developers of all experience levels can act on immediately.
- Maintain developer trust via deterministic verification and human-in-loop remediation.

Success Metrics (12 Months)

- Time-to-detection (median) for confirmed critical exploit paths ≤ 24 hours after code push.
- False-positive rate for confirmed exploits ≤ 5% (measured after runtime validation).

- First meaningful finding produced within 10 minutes of repo onboarding for ≥ 80% of repos (including simple ones).
- Developer adoption: 500 active repos within 12 months, distributed across all repo sizes.
- Customer retention: 70% monthly retention for paid accounts after 6 months.
- Average time-to-fix after PR report ≤ 48 hours (paid customers using suggested patch workflow).
- Free tier conversion rate: ≥ 10% of free users upgrade within 90 days.

Target Users & Personas

Primary Users

- Solo / small-team backend developers ("vibe coders") shipping API-first apps of any complexity.
- Beginners and bootcamp graduates shipping their first real-world APIs.
- Early-stage startup engineering teams with no full-time AppSec.
- Software houses and agencies building client APIs (simple CRUD to complex multi-tenant).

Personas

Maya — solo founder & dev: Builds a two-route SaaS using Next.js/Express. Needs immediate, plain-English security feedback in CI without needing to understand security jargon.

Ravi — engineering lead at seed startup: Wants continuous red-team coverage on a growing, complex API that she can trust and include in PR process without noise.

Ammar — dev at small agency: Needs quick proofs to show clients and to harden both simple client portals and multi-tenant APIs.

Dev — bootcamp grad, first job: Just deployed their first Express API to Heroku. Has no security knowledge. Needs AART to explain what went wrong in plain language and show exactly how to fix it.

Problem Statements

- Developers lack attacker-mode thinking and miss chained access-control issues — at all skill levels.
- Static scanners produce noisy, unactionable findings and are often misconfigured for simple repos.
- Startups and solo devs can't afford manual red teaming or a dedicated AppSec hire.

- Simple repos are ignored by enterprise security tools, leaving the long tail of small apps completely unprotected.
- Security findings are often unreproducible or lack clear minimal fixes that junior developers can understand.

Value Proposition

- Universal: Works on any Node.js/Express repo regardless of size, complexity, or the developer's experience level.
- High signal: Only produce verified exploit chains — no noise, no guesswork.
- Instantly actionable: Plain-English explanations + copy-pasteable curl commands + minimal patch diffs.
- Fast & continuous: Run micro-red-team checks on PRs and commits, even for tiny repos.
- Trustworthy: Deterministic engine is authoritative; LLMs assist but cannot claim exploit success.
- Approachable: PR comments read like a senior engineer explaining a bug, not a security scanner output.

Key Features

MVP (4–6 Months)

Universal Repo Support

- Lightweight fast-path scanner for simple repos (1–10 routes): produces near-instant heuristic findings without full graph/sandbox pipeline.
- Full graph + symbolic pipeline for medium/complex repos (10+ routes, multiple roles/models).
- Auto-detection of repo complexity at onboarding; routes appropriate pipeline automatically.
- Graceful degradation: even if symbolic engine finds nothing, static heuristic layer always reports something meaningful.

Core Engine

- Repo ingestion + deterministic AST-based route & middleware extractor (Node.js/Express).
- Attack surface graph builder (routes, middleware, models, roles).
- Symbolic-execution-lite module to infer ownership checks and reachability.
- IDOR / Broken access control / Horizontal privilege escalation simulation engine.
- Ephemeral sandbox runner (Docker-based) to validate high-confidence candidates.

Developer Experience

- CI/CD integration as a GitHub App: post plain-English PR comments with findings.
- Plain-language severity descriptions ("Anyone can read any user's invoice" not "IDOR CWE-639").
- Report generator: structured JSON + human narrative + reproducible curl steps + suggested minimal patch (LLM-assisted).
- "Why this matters" section in every finding, tailored to the repo's context.
- Threat memory: store exploit patterns and app fingerprints.

Phase 2 (6–12 Months)

- Auto-retest & regression checks after patch PRs.
- Patch-as-PR capability (developer approval required).
- Support for GraphQL, Fastify, and basic Django/FastAPI scanning.
- Confidence-based runtime feedback expansion.
- User dashboard and executive view (business risk summaries).
- Security score / "health grade" per repo visible at a glance.

Phase 3 (12–24 Months)

- Multi-service chaining within single repo or multi-repo applications.
- Multi-tenant / SaaS isolation checks.
- Compliance mapping (SOC2/ISO) and ATT&CK alignment.
- Integrations: Slack/MS Teams, Jira, CI platforms (GitLab, Bitbucket).
- Interactive security education mode: explain concepts in context of the developer's own code.

Product Constraints & Guiding Principles

- Safety-first: Never run attacks against production systems. All runtime verification occurs in ephemeral, isolated sandboxes.
- Universal coverage: Every repo — no matter how small — receives at minimum a heuristic-level scan. No repo is "too simple" to analyze.
- Deterministic authority: Deterministic engine always controls final exploit feasibility decisions.
- Human-in-loop: No automatic production changes; developers approve fixes and PRs.
- Approachability: All user-facing language is jargon-free unless the user configures advanced mode.
- Minimal surface early: Target Node.js/Express and JWT-auth patterns first, then expand.

Detailed User Flows

1. Onboard Repo (First-Time)

1. Developer installs GitHub App and grants repo access.
2. AART clones repo in ephemeral worker, detects complexity tier (simple / medium / complex).
3. Simple repos (≤10 routes): fast-path heuristic scan runs immediately (≤3 minutes) and returns first findings.
4. Medium/complex repos: full AST parse, graph build, symbolic evaluation runs (≤10 minutes).
5. System returns "App Fingerprint" report: stack, auth type, notable models, security health grade.

2. PR Check (CI)

1. Developer opens PR. GitHub App triggers AART check.
2. Complexity-appropriate pipeline runs (fast-path or full graph).
3. For high-confidence candidates, sandbox is spawned, simulation executed.
4. AART posts PR comment: plain-English summary, severity, proof (sanitized curl), suggested patch, confidence score.
5. Developer chooses action: apply suggestion locally, accept AART-created PR draft, or mark as false positive.

3. Patch Lifecycle

1. Developer applies fix (or accepts AART PR).
2. AART re-runs simulation in sandbox.
3. If exploit fails, AART marks it resolved and updates threat memory.

Technical Architecture (Summary)

Component Overview

- Ingestion Worker: clones repo, normalizes code, detects complexity tier, extracts ASTs.
- Complexity Router: routes repo to fast-path heuristic scanner or full graph + symbolic pipeline.
- Heuristic Scanner: lightweight static checks for simple repos (missing auth middleware, obvious IDOR patterns).
- Graph Builder: constructs attack surface graph (routes, middleware, models) for medium/complex repos.
- Symbolic Lite Engine: symbolic variable tracking and constraint evaluation.

- LLM Reasoner: constrained calls for semantic inference & patch drafts (schema-enforced).
- Attack Planner: generates candidate attack plans from graph + templates.
- Sandbox Runner: Docker-based isolated environment; seeds DB with synthetic users and data.
- Execution Orchestrator: sends crafted requests, captures evidence.
- Threat Memory DB: stores historical exploits, app fingerprints, fix patterns across all repo tiers.
- CI Integrations & GitHub App: webhook handlers and PR comments.
- Dashboard / API: user interface and endpoints for results and controls.

Complexity Tiers

Tier	Criteria	Pipeline	Target Time
Simple	≤10 routes, 1 role, minimal models	Heuristic fast-path scanner	< 3 min
Medium	10–50 routes, 2–3 roles, multiple models	Graph + symbolic + selective sandbox	< 8 min
Complex	50+ routes, multi-role, multi-tenant	Full graph + symbolic + sandbox + LLM	< 15 min

Data & Model Designs

- AppGraph: nodes (route, middleware, model, role), edges with metadata (ownership fields, auth checks). Populated even for simple single-file repos.
- ExploitRecord: {id, appFingerprint, repoTier, route, exploitType, evidence, confidence, status, plainLanguageSummary}
- ThreatMemory: mapping of appFingerprint -> recurring exploit patterns, preferred remediation templates, repo tier.

Security, Privacy & Compliance

- Sandboxes run in isolated network namespaces with no external access by default.
- Secrets handling: never store production secrets. If accidentally included, flag and notify owner but never use them in sandbox.
- Data retention: artifacts and exploit evidence retained per user-defined windows (30/90/365 days).
- GDPR/Privacy: support export/deletion of customer data; document data processing.
- Compliance options: enterprise-only on-prem/VPC deployment for customers requiring no external data flow.

Non-Functional Requirements

- Performance: Simple repos analyzed in < 3 minutes; medium repos < 8 minutes; complex repos < 15 minutes.
- Universal coverage: No repo is rejected or skipped due to size or simplicity.
- Scalability: Autoscale sandbox workers based on queue depth.
- Reliability: 99.5% availability for the core API.
- Cost control: enforce caps on sandbox runtime and resource usage per account.
- Explainability: every confirmed exploit must include deterministic evidence and a plain-language explanation accessible to junior developers.

Governance of AI Usage

- All LLM outputs must conform to a strict JSON schema; the deterministic engine validates them.
- LLMs can propose patches and interpret ambiguous code; they cannot mark an exploit as confirmed.
- LLM-generated patch suggestions are always labeled as "draft" and require developer approval.
- Maintain an audit log for all LLM interactions.

Go-to-Market & Pricing

Free Tier

1 repo, limited monthly sandbox minutes, PR checks with non-blocking comments. Designed specifically to give solo devs and students immediate value on their first repo.

Starter

Up to 5 repos, increased sandbox minutes, PR blocking optional, email support.

Startup

10-25 repos, team features, Slack integration, priority support.

Enterprise

Unlimited repos, on-prem/VPC deployment, compliance features, SLA & dedicated onboarding.

Acquisition Channels

- Developer community content (blogs, demo videos showing AART finding bugs in tiny repos)
- GitHub Marketplace / GitHub App listing
- Bootcamp and learn-to-code community partnerships
- Partnerships with dev platforms and startup accelerators
- Targeted outreach to dev agencies

Roadmap & Milestones (12 Months)

- Month 0-2: Core team, repo ingestion, complexity router, heuristic fast-path scanner, parser, graph builder prototype.
- Month 3-4: Symbolic-lite engine, IDOR detection, attack planner, validate fast-path on simple repos.
- Month 5-6: Sandbox runner + PR integration, plain-language report generator, MVP launch (closed alpha).
- Month 7-9: Expand attack families, LLM patch drafts, public beta targeting indie devs and bootcamp grads.
- Month 10-12: Hardening, dashboard with security health grade, pricing, go-to-market, early enterprise pilots.

Risks & Mitigations

- Risk: Simple repos produce no findings, making AART seem useless for the mass market. Mitigation: Heuristic fast-path layer always produces at least advisory-level findings; even "no critical issues found" is surfaced with proactive tips.
- Risk: False positives reduce trust. Mitigation: Runtime validation required for confirmed findings; confidence thresholds; easy developer feedback loop.
- Risk: Sandbox escapes or accidental production impact. Mitigation: Strict network isolation, resource limits, no production credentials.
- Risk: LLM hallucinations produce poor patches. Mitigation: Deterministic validation in sandbox; patches labeled as "draft".
- Risk: High operating cost from heavy sandbox usage on simple repos. Mitigation: Fast-path scanner avoids sandbox for simple/obvious patterns; cost quotas per account.

KPIs to Track

- Repos onboarded by tier (simple / medium / complex)
- First-finding time by tier
- Active checks per repo

- Confirmed exploit count and false positive rate
- Average compute minutes per confirmed exploit
- Developer response time to AART suggestions
- Free-to-paid conversion rate
- MRR growth & churn

Acceptance Criteria (MVP)

- The system produces a meaningful security finding on a minimal Node.js/Express app (single route + single model) within 3 minutes of ingestion.
- The system can analyze a standard medium-complexity Node.js/Express repo and produce an AppGraph within 8 minutes.
- The symbolic-lite engine detects at least three classes of exploit candidates (IDOR, horizontal escalation, mass assignment).
- The heuristic fast-path layer produces advisory findings even when the symbolic engine finds no candidates.
- The sandbox runner executes and validates candidate exploits safely and reproducibly.
- The GitHub App posts plain-English PR comments with evidence, suggested minimal patch, and confidence score.
- Threat memory stores exploit history and influences subsequent checks.

Appendix — Example PR Comment (Sanitized)

Short executive summary: "Anyone can read anyone else's invoice — here's how."

Plain-language explanation: "Your GET /invoices/:id endpoint doesn't check that the invoice belongs to the user making the request. That means user A can read user B's invoice just by changing the number in the URL."

Confidence: 0.92 (symbolic 0.86 + runtime validation PASS).

Repro steps: authenticated curl (token redacted) + sanitized request + expected response.

Suggested fix (minimal): add ownership check in controller (LLM-drafted diff link).

Test after fix: "Re-run checks" button.