audited.xyz

Date	Commit
Mar 21, 2026 09:27 UTC	`main`
	558 files, 89,196 lines

Summary

Audited.xyz is a Ruby on Rails 8 platform that provides automated security audit services for smart contracts and web applications. The system accepts code via GitHub integration, direct file upload, or on-chain contract address, then orchestrates multi-pass security analysis through Claude AI using a chunked audit pipeline with domain-specific prompt templates spanning Solidity, Rails, Python, Rust/Solana, and frontend frameworks. The platform implements a full commercial workflow including tiered pricing with Stripe and on-chain USDC payment verification, referral commissions with both fiat (Stripe Connect) and crypto (smart contract) withdrawal paths, Ethereum Attestation Service (EAS) on-chain attestation of audit results, and an automated fix loop that applies AI-generated patches via GitHub pull requests. The architecture comprises UUPS-upgradeable Solidity payment contracts, a Rails backend with Solid Queue for background job orchestration, ActionCable WebSocket channels for real-time status updates, Stimulus.js controllers for client-side interactivity, and PostgreSQL with database-level integrity constraints as the persistence layer.

The codebase demonstrates a defense-in-depth security architecture applied consistently across all layers. Input handling employs SSRF protection with DNS rebinding prevention and IP blocklists in document fetching services, ZIP bomb detection via compression ratio analysis, symlink and path traversal defenses with TOCTOU-aware double-checks in code extraction, and prompt injection sanitization for the LLM integration pipeline. Cryptographic operations follow current best practices: API keys are stored as BCrypt hashes with constant-time comparison via `ActiveSupport::SecurityUtils.secure_compare`, webhook signatures use HMAC with timestamp-based replay protection and secret rotation support, the Ethereum signing implementation enforces EIP-2 low-s normalization and EIP-155 replay protection, and GitHub OAuth tokens are stored using Rails encrypted attributes. Financial integrity is protected through PostgreSQL advisory locks on payment completion flows, atomic SQL credit operations with database-level non-negative check constraints, and unique indexes on payment transaction hashes to prevent double-processing. The ActionCable layer implements session version tracking with cache-based invalidation propagation, periodic revalidation at configurable intervals, origin validation with strict host/scheme/port matching, and database-level row locking for admin channel subscriptions to prevent TOCTOU race conditions.

The overall security posture of audited.xyz is strong, reflecting multiple rounds of prior security review and deliberate remediation. No critical, high, or medium-severity vulnerabilities were identified across the full scope of this engagement, which covered the complete application stack from Solidity smart contracts and database migrations through the Rails service layer, controllers, models, background jobs, WebSocket channels, JavaScript controllers, and all view templates. The Solidity payment contracts employ OpenZeppelin's battle-tested libraries for upgradeability, reentrancy protection, and safe token transfers, with proper `_disableInitializers()` in constructors and owner-restricted upgrade authorization. The client-side JavaScript consistently uses safe DOM manipulation via `textContent` and `createElement` rather than `innerHTML`, validates redirect URLs through a centralized same-origin helper with explicit domain allowlisting, and loads external dependencies with Subresource Integrity verification. The self-audit subsystem and comprehensive regression test suite — which includes explicit test cases for every

previously discovered vulnerability — demonstrate an organizational commitment to continuous security validation.

Conclusion

The audited.xyz codebase exhibits a high degree of security maturity across its full technology stack. The service layer's treatment of the LLM integration pipeline as an untrusted boundary — with content sanitization, finding injection marker detection, and structured output parsing — reflects a sophisticated understanding of the risks inherent in AI-assisted code analysis. The Rails view layer maintains strict discipline in output encoding, with all audit view templates, admin templates, and every other ERB file consistently using Rails' default auto-escaping, reserving `raw` exclusively for Commonmarker-rendered markdown configured with `unsafe: false` to strip embedded HTML. The database schema's evolution over 90+ migrations shows consistent security-aware design decisions, from the transition away from soft-delete to prevent sensitive data retention, to the addition of compound performance indexes and database-level constraint enforcement for financial operations.

The most significant architectural risk areas are the AI-generated fix application pipeline, where the platform applies LLM-produced code patches to customer repositories via GitHub pull requests, and the centralized admin control over the UUPS-upgradeable payment contracts, which currently rely on a single-owner key rather than a timelock or multisig. The fix loop pipeline includes a sensitive path allowlist to prevent modifications to security-critical files, though expanding this allowlist to cover additional configuration and deployment paths would strengthen the defense-in-depth posture. The on-chain payment system's admin centralization is appropriate for the current scale of operations but represents a trust assumption that should be revisited as transaction volume grows. Neither of these areas presents a directly exploitable vulnerability under current conditions, but both warrant ongoing attention as the platform scales.

Audited.xyz is well-prepared for continued production operation. The application enforces SSL with HSTS preloading, implements multi-tier Rack::Attack rate limiting across general requests, file uploads, API endpoints, and authentication flows, validates required environment variables at boot to prevent misconfigured deployments, and runs as a non-root user in its containerized deployment. The comprehensive regression test suite covering previously identified vulnerabilities provides strong assurance against reintroduction of fixed issues. The recommended path forward is to maintain the existing practice of security regression testing for every new finding, consider governance decentralization for the on-chain payment contracts as the protocol matures, and continue periodic review of the prompt sanitization pipeline as new LLM interaction patterns are introduced through the chunked and multi-pass audit workflows.