DCA.fund

audited.xyz
by zack.eth

Date

Mar 22, 2026 00:21 UTC

Commit

`main`

382 files, 53,370 lines

Summary

DCA.fund is a dollar-cost averaging protocol deployed on Ethereum and Base that enables automated periodic token swaps between USDC, cbBTC, and WETH via Uniswap V3, with idle user deposits generating yield through Aave V3 lending pools. The system architecture centers on a UUPS upgradeable proxy contract delegating business logic to eight external Solidity libraries sharing namespaced storage via ERC-7201, supported by peripheral contracts for swap execution, balance computation, reward distribution, referrals, and read-only views. Beyond the on-chain protocol, the codebase encompasses a React/TypeScript frontend with Privy and MetaMask wallet integration, fiat onramp flows through Coinbase Pay, Stripe, and MoonPay, a lightweight Express.js backend serving as a testnet faucet and onramp session proxy, a Cloudflare Worker reverse proxy for production routing, and Hardhat-based deployment and testing infrastructure spanning multiple networks. The protocol also integrates with CoW Protocol for off-chain settlement, enabling MEV-protected order execution alongside its direct Uniswap V3 swap path.

The codebase demonstrates deliberate, layered security design across both on-chain and off-chain components. The smart contracts employ OpenZeppelin's `ReentrancyGuardUpgradeable` on all state-modifying entry points, enforce `onlyOwner` access control for administrative functions including upgrade authorization, and use `_disableInitializers()` to prevent implementation contract takeover. Uniswap V3 swaps are protected by quoter-derived slippage bounds with a five-minute deadline buffer, and the `saveMyBalanceCheckpoint` function was hardened to recompute unrealized balances from scratch rather than trusting user-supplied pagination state. On the frontend, third-party payment integrations validate `postMessage` origins against strict allowlists, the MoonPay HMAC signing endpoint restricts URL targets to the legitimate payment domain, wallet addresses are normalized via `ethers.getAddress()` throughout, and the runtime configuration system isolates server secrets from client-exposed values by restricting injection to `VITE_*`-prefixed environment variables. The backend faucet implements signature verification with timestamp expiry and an in-memory pending-claims guard against time-of-check/time-of-use races.

The audit of the full DCA.fund codebase identified no critical, high, or medium-severity vulnerabilities across all fourteen sections encompassing smart contracts, frontend, backend, deployment scripts, configuration, and infrastructure. The protocol's fund handling, access control model, and arithmetic are sound, with interest accrual correctly tracking Aave's normalized income via scaled-balance accounting and the referral system enforcing one-time binding before first deposit. The primary risk surface is the standard centralization inherent in owner-controlled UUPS upgradeability, which subsumes all other administrative privileges, and the protocol's dependency on external liquidity sources — Uniswap V3 quoter availability and Aave lending pool health — for core withdrawal and swap operations. The absence of exploitable vulnerabilities in the smart contract layer, combined with consistent application of established security patterns across all system boundaries, reflects a codebase that has been developed with security as a first-order design constraint.

Conclusion

The DCA.fund codebase exhibits strong security maturity across its full stack. The Solidity contracts follow a disciplined library-based decomposition that keeps the main proxy contract within bytecode limits while maintaining a coherent shared storage model through ERC-7201 namespaced slots — a pattern that reduces upgrade risk and simplifies reasoning about state invariants. Code quality is consistent from the smart contract layer through the frontend: input validation occurs at every system boundary, authorization decisions are uniformly delegated to on-chain logic rather than replicated in client code, and configuration management properly separates testnet from mainnet parameters with strict TypeScript typing that prevents misconfiguration at compile time. The test suite demonstrates solid coverage of core protocol invariants including deposit and withdrawal flows, DCA and TWAP order lifecycles, CoW settlement, upgrade mechanics, and shutdown behavior, using mainnet fork testing for realistic conditions.

The most significant architectural consideration for the protocol in production is its dependency on external DeFi infrastructure during user-critical operations. The mandatory interest-to-USDC conversion during cbBTC and WETH withdrawals creates a coupling between withdrawal availability and Uniswap V3 quoter responsiveness — a path that could impede user access to principal during periods of network congestion or liquidity fragmentation, precisely when withdrawal demand is highest. While no funds are at risk of theft through this path, availability under stress is a meaningful concern for a protocol holding user deposits. The owner's ability to reconfigure CoW settlement addresses, vault relayer approvals, and swap parameters through administrative functions is standard for managed DeFi protocols but represents a trust assumption that users should understand.

DCA.fund is suitable for mainnet deployment in its current state. The smart contract architecture is well-constructed, the security control surface is comprehensive, and no vulnerabilities were identified that would enable fund loss or unauthorized access. Prior to deployment, the team should evaluate whether the withdrawal path can be made resilient to temporary Uniswap quoter unavailability — for example, by allowing users to withdraw their principal without the interest conversion when the swap path fails — to ensure the protocol maintains availability guarantees under adverse market conditions. A focused re-audit of any modifications to the withdrawal and swap interaction would be appropriate to verify no regressions are introduced.