Security Audit Report

Audit prepared for

LiquidLiberty

Updated on

July 2, 2026

TABLE OF CONTENTS

Technical Summary 01

Scope of Audit 02

Methodology 03

System Architecture Overview 04

Code Quality & Best Practices 05

Findings & Vulnerability Analysis 06

Detailed Findings 07

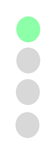
Simulation & Testing 08

Integration & Economic Safety 09

Final Statement & Conclusion 10

About SPYWOLF 11

Disclaimer 12



TECHNICAL SUMMARY(1)

Project Overview

Liquid Liberty Protocol Phase 1 is a standalone deployment of the protocol's "vault flywheel." It is composed of:

- **Core contracts:** LibertyVault, a stablecoin-backed vault that mints/burns a vault token (dMKT) whose per-token redemption price is monotonically non-decreasing by construction, and LibertyVaultToken, the ERC-20 + ERC-2612 share token that the vault exclusively mints and burns.
- **Off-chain signer service:** an EIP-712 signing service (apps/core-api) that issues the trustedSigner mint/burn payloads every user operation requires, gated by a shared-secret Bearer-token auth layer.
- **Proxy layer:** server-side Netlify Functions (apps/web/netlify/functions/) that hold the signer API key and inject it on behalf of the browser SPA.

The vault is internally priced (price = totalCollateral / supply), retains 100 bps of spread per operation as permanent backing, is paired against a third-party DEX (PulseX V2), and is kept on peg by an off-chain arbitrage bot. Owner powers are limited to a binary spread toggle, treasury wiring, and a one-way renounce; the trusted signer is immutable and bootstrap is one-shot.

Audit Objective

To perform a **full technical security review** of the in-scope LibertyVault contracts and the EIP-712 signer/auth surface, identifying exploitable errors, logic inconsistencies, broken invariants, and unsafe assumptions in the flow of funds. The focus is **error and exploit prevention**, specifically the safety of user collateral and the integrity of the vault's price and accounting invariants, not governance or trust decentralization, which were explicitly out of scope per the engagement.

Audit Scope

Category	Components	Focus Areas
Core Vault	LibertyVault.sol, LibertyVaultToken.sol	EIP-712 mint/burn, spread and collateral accounting, price monotonicity, share-inflation resistance, solvency and conservation, renounce and bootstrap semantics, reentrancy
Signer Service	core-api/src/domains/signatures/, signer-config.service.ts	EIP-712 domain and typehash correctness, nonce and deadline handling, signing-key custody, signature-issuance controls
Auth & Proxy	shared/services/auth.service.ts, apps/web/netlify/functions/	Timing-safe key comparison, fail-closed configuration, server-side secret handling, caller authorization

TECHNICAL SUMMARY(2)

Code reviewed: in-scope Solidity (LibertyVault ~440 LoC, LibertyVaultToken ~49 LoC) plus the ~340 LoC TypeScript signer, auth, and proxy surface, for roughly 830 LoC total. LibertyArbRouter, deploy scripts, mocks, the dApp UI, and the off-chain bots were reviewed for context only and are not part of paid scope.

Environment:

- Solidity 0.8.20, optimizer enabled (200 runs); EVM version not pinned in the audited tag, pinned to Paris in remediation (see F-05)
- Chain: PulseChain mainnet, with bridged DAI as collateral and PulseX V2 as the DEX
- TypeScript signer service (Node, ethers v6) deployed as Netlify Functions
- In-repo Hardhat suite (122 passing) reviewed; SpyWolf supplemented this with an independent exact-integer model of the vault arithmetic for invariant and adversarial fuzzing

Overall Assessment

The Liquid Liberty Phase 1 codebase is **well-engineered, disciplined, and thoroughly documented.** The architecture deliberately closes the attack classes that most often compromise share-based vaults. Pricing reads an internal totalCollateral accumulator rather than the live token balance, which **neutralizes the classic ERC-4626 first-depositor and donation inflation attack.** The trusted signer is immutable, which removes the owner-to-signer escalation path. Bootstrap is one-shot, which closes the post-launch dilution vector. And every mint and burn is bound on-chain to msg.sender, which keeps the blast radius of any signer or API-key compromise limited to commercial (arb-revenue) loss rather than theft of user funds.

Our review combined line-by-line manual analysis with an independent reimplementation of the vault's integer arithmetic, exercised across several hundred randomized operation sequences plus targeted edge-case and value-conservation tests. The vault held its core invariants throughout. It stayed fully solvent, holding collateral greater than or equal to its accounted collateral at all times. Value was conserved to the wei, with no sequence allowing an attacker to extract more than they deposited. And the redemption price stayed monotonic everywhere except a single boundary condition that we document in the findings.

No Critical or High-severity issues were identified. The initial review found two Medium-severity items and a small set of Low and Informational items. All eight findings (F-01 through F-08) were remediated by the team in the patched revision v0.1.1-audit-patched and verified by SpyWolf. Each is marked FIXED in the findings section.

The two Medium-severity findings, both now resolved:

1. Full-redemption price reset (supply to zero). If the entire vault-token supply were burned to zero, getCurrentPrice() would reset to its 1e18 sentinel, breaking the stated monotonically non-decreasing invariant, and the spread retained on the final burn would be left backing zero supply for the next minter to capture. Resolved by minting a permanent dead-share floor at bootstrap and gating the bootstrap branch on currentSupply == 0 only, so supply can never reach zero and the price never resets.
2. Auth gate nullified by the public proxy. The Netlify mint and burn proxies injected the secret SIGNER_API_KEY for any caller and served Access-Control-Allow-Origin: *, so the "only our UI and bots" control could be bypassed by anyone reaching the public proxy URL. The impact was commercial, not fund loss, since the on-chain msg.sender binding bounds it. Resolved with an origin allowlist, per-wallet and per-IP rate limiting, and restricted CORS on both the proxy and core-api.

SCOPE OF AUDIT(1)

The audit covered the in-scope contracts and the off-chain signing surface delivered in the LBRTYphase1-audit handoff repository, frozen at tag v0.1.0-audit:

1. **Core Contracts** from contracts/
    - LibertyVault.sol: the vault itself, covering EIP-712 signature-gated mint and burn, spread and collateral accounting, the price formula, treasury wiring, and renounce and bootstrap semantics.
    - LibertyVaultToken.sol: the ERC-20 + ERC-2612 share token, mintable and burnable only by the vault.
2. **Signer Service** from apps/core-api/
    - src/domains/signatures/: the EIP-712 signing service, including the mint and burn handlers, the signature service, and the domain and type definitions.
    - src/shared/services/auth.service.ts: the Bearer-token auth gate.
    - src/shared/services/signer-config.service.ts: signing-key discovery and custody.
3. **Proxy Layer** from apps/web/netlify/functions/
    - request-mint-signature.ts and request-burn-signature.ts: the server-side functions that hold the shared secret and forward signing requests to the core API.

Excluded Components

- LibertyArbRouter.sol, the stateless arbitrage aggregator, was provided as context and held out of paid scope. It holds no user funds, pulls only from msg.sender, and its routes are fixed at construction.
- Deploy and operational scripts (scripts/deploy-*.js, oa-mint-for-lp.js, seed-dex.js, and related helpers) were read for context to understand how the audited contracts are instantiated, but were not audited for security. We did spot-check deploy-mainnet.js for initialization ordering at the project's invitation.
- Test mocks (contracts/mocks/) and the in-repo Hardhat test suite were reviewed to understand coverage but were not themselves treated as audit targets.
- The dApp front end (apps/web/src/), the off-chain arbitrage bot, the load-test bot, the dashboard, and the trading simulator were out of scope.
- Centralized governance and owner key trust were not assessed for trust minimization, per the scope agreement.

Code Inventory

Repository Area	Files	Lines of Code	Language
Core Contracts	2	506	Solidity
Signer Service	9	679	TypeScript
Proxy Layer	2	167	TypeScript
Total	**13**	**1,352**	-

Total: 1,352 lines reviewed manually across 13 in-scope files. The signer service's shared response, validation, and logger utilities were reviewed as direct dependencies of the in-scope handlers but are not counted above. Of this surface, roughly 340 lines on the signer side and the full vault logic represent the security-relevant code; the remainder is type definitions, constants, NatSpec, and barrel exports. Review was supplemented with pattern-based static analysis and an independent exact-integer reimplementation of the vault arithmetic used for invariant and adversarial fuzzing.

SCOPE OF AUDIT(2)

Audit Assumptions

- Team-controlled keys are trusted. The owner key, the trusted signer key, and the SIGNER_API_KEY are assumed to be held by the operator, and centralization is not in scope.
- The underlying collateral is a trusted, standard ERC-20. Phase 1 uses bridged DAI, assumed to be non-rebasing and free of transfer fees and callback hooks.
- No malicious governance or owner action is considered, consistent with the engagement's exclusion of trust-minimization analysis.
- The PulseChain environment behaves equivalently to EVM mainnet at the Paris hard fork level.
- The vault is operated alongside a seeded PulseX V2 liquidity pool, which is the intended Phase 1 topology.
- Bridge solvency and depeg behavior of bridged DAI are acknowledged as inherited external risks and are outside the contract review.

Repository Handoff

The review was performed against a private, audit-only repository delivered with read and triage access after a mutual NDA. Findings in this report reference the frozen commit at tag v0.1.0-audit. Any hotfixes applied during the engagement are expected to land on a separate branch so the audited tag itself is not moved.

METHODOLOGY(1)

Audit Methodology Overview

A hybrid approach was used, combining **manual code review**, **static pattern analysis**, and **dynamic simulation**. Because the protocol's safety rests almost entirely on the vault's integer arithmetic and its signature flow, the simulation effort was weighted toward proving the economic invariants hold under adversarial sequences, not just confirming the happy path.

1. Manual Review

Each in-scope file was reviewed line by line to verify:

- State transition correctness across mint, burn, bootstrap, and the treasury paths
- Access control and function visibility, including the vault-only mint and burn on the share token
- Arithmetic accuracy, with attention to rounding direction and the use of internal accounting versus live token balances
- Spread retention and collateral accounting on every value-moving path
- External call handling and ordering, and the placement of the reentrancy guard
- Signature construction, nonce handling, deadline enforcement, and EIP-712 domain and typehash agreement between contract and signer
- Edge cases around empty supply, zero amounts, and the first deposit

2. Static Analysis

Pattern-based scanning was used to surface code sections that warrant closer manual inspection. Patterns scanned:

- ecrecover and ECDSA usage, DOMAIN_SEPARATOR, typehash definitions, and nonce increments
- transfer and transferFrom without SafeERC20, and any reliance on returned balances instead of deltas
- Internal accounting variables (totalCollateral, totalSupply) versus balanceOf reads in pricing math
- Reentrancy surfaces and the presence of nonReentrant on user-facing entry points
- Division and rounding operations in the mint, burn, and price calculations
- On the off-chain side: timing-safe comparison usage, environment-variable fallbacks, CORS configuration, and error responses that echo internal detail

3. Dynamic Testing & Simulation

The in-repo Hardhat suite (122 passing tests) was reviewed for coverage and correctness of its assertions. SpyWolf then built an independent exact-integer model of the vault arithmetic, reproducing Solidity 0.8.20 floor division and checked over- and underflow behavior, and exercised it under conditions the in-repo tests do not reach. Simulations covered:

- Price monotonicity across long randomized mint and burn sequences
- Vault solvency, confirming held collateral never falls below accounted collateral
- Share-inflation and donation attempts against a first or early depositor
- The supply-to-zero boundary and its effect on the price formula
- Mint-then-burn round trips, to confirm a round trip is never profitable for the caller
- Signature replay, nonce reuse, and cross-type reuse between mint and burn payloads
- Reentrancy attempts through the underlying token on the mint and burn paths
- Auth-gate behavior across missing, malformed, wrong-length, and correct API keys

METHODOLOGY(2)

4. Fuzz & Invariant Testing

The model was driven with randomized inputs and deterministic seeds so results are reproducible. The properties asserted were:

- **Liquidity conservation:** total collateral in equals total collateral out plus collateral still backing outstanding supply, to the wei
- **Price monotonicity:** getCurrentPrice() never decreases across any sequence, with the single boundary exception recorded in the findings
- **Over-collateralization:** outstanding supply valued at the current price never exceeds total collateral
- **No value extraction:** across several hundred randomized sequences, no attacker-controlled actor was able to withdraw more than it deposited
- **Nonce integrity:** per-user nonce strictly increments, and no signature is replayable

5. Tools & Framework

Category	Tool	Purpose
Manual Review	Reviewer-led line-by-line analysis	Logic, access control, accounting, signature flow
Static Analysis	Pattern and rule-based scanning (regex, Python)	Surface unsafe patterns and high-risk code sections
Contract Testing	Hardhat (in-repo suite)	Review of the project's 122 existing tests
Simulation	Independent exact-integer model (Python)	Invariant and adversarial behavior under randomized sequences
Reporting	Markdown	Traceable documentation of findings

6. Environment Configuration

Parameter	Value
Solidity Compiler	0.8.20
Optimizer	Enabled, 200 runs
Contract Test Framework	Hardhat (in-repo, 122 passing)
Simulation	Exact-integer model, deterministic seeds
Off-chain Runtime	Node.js, ethers v6 (signer service review)

Conclusion of Methodology Section

The combined manual, static, and simulation approach gives coverage of both functional correctness and exploit resistance. Weighting the simulation work toward the vault's economic invariants, rather than only its function-level behavior, is what let the review confirm that the accounting stays solvent and conservative under adversarial sequences, and isolate the one boundary condition where the price formula departs from its stated invariant.

SYSTEM ARCHITECTURE OVERVIEW(1)

High-Level System Description

Liquid Liberty Phase 1 is a deliberately small system. On-chain it is a single vault and its share token; off-chain it is a signing service and a proxy that authorize every user operation. There is no factory, no plugin layer, and no peer contracts in this phase, so the entire trust surface comes down to two questions: does the vault account for collateral correctly, and who is allowed to obtain a signature.

Layer	Purpose	Key Components
Core (on-chain)	Holds collateral, mints and burns the share token, computes price	LibertyVault, LibertyVaultToken
Signer (off-chain)	Issues the EIP-712 mint and burn payloads each operation requires	core-api signature service, auth.service, signer-config
Proxy (off-chain)	Holds the API secret and forwards browser requests server-side	request-mint-signature, request-burn-signature

Collateral is custodied only by the vault. The share token (LibertyVaultToken) can be minted and burned by one address, the vault, which is set as its owner at construction. The off-chain layers never custody funds; they only decide whether a signature gets issued.

Mint and Burn Lifecycle

The vault has no permissionless write path. Every mint and burn requires a fresh signature from the trustedSigner, and the flow is the same in both directions:

1. The client fetches the user's current on-chain nonce and requests a signature from the proxy.
2. The proxy injects the SIGNER_API_KEY and forwards to the core API, which checks the key, then signs the (user, amount, nonce, deadline) payload with a 30-minute server-set deadline.
3. The user submits mint() or burn() on the vault with that signature.
4. The vault rebuilds the digest using msg.sender and the on-chain nonce, recovers the signer, and requires it to equal trustedSigner.
5. On mint, the user pays amount + spread, receives shares priced on amount, and the full inflow is retained as collateral. On burn, the user receives the underlying value of their shares minus spread, and the retained spread stays as backing.

Two structural properties fall out of this flow and are worth keeping in mind for the findings:

- The digest binds to msg.sender. A signature issued for one address cannot be used by another, and the caller always pays from and receives to their own balance. This is what keeps any signer or key compromise limited to commercial impact rather than theft.
- Price is totalCollateral / supply, computed from internal accounting variables rather than the vault's live token balance. This is what neutralizes the standard donation and first-depositor inflation attack, since a direct token transfer to the vault does not move the price.

SYSTEM ARCHITECTURE OVERVIEW(2)

Trust Boundaries

Boundary	What it controls	Notes
Vault custody	Collateral and share supply	Only the vault moves collateral; only the vault mints or burns shares
On-chain signature check	Whether a mint or burn executes	Enforces signer identity and nonce; cannot enforce who requested the signature off-chain
Off-chain auth gate	Who can obtain a signature	Shared-secret Bearer token; the commercial gate on signature issuance
Proxy	Holds the secret for the browser	Server-side; the browser never sees the key

The critical thing this layout makes explicit is the split between the on-chain check and the off-chain gate. The contract can prove a signature came from trustedSigner, but it has no way to know who asked the signer for it. That separation is by design, and it is the reason the auth gate is a commercial control rather than a protocol-safety control. Finding M-02 concerns the strength of that off-chain gate, and finding M-01 concerns a boundary condition in the price formula that the custody layer relies on.

Design Observations

- The on-chain surface is minimal and the absence of a recovery or pause path is intentional, which trades operational flexibility for a smaller attack surface.
- Safety leans heavily on the off-chain signer being available and its key being held securely, since there is no on-chain rotation path once deployed.
- The same totalCollateral accumulator that defends against inflation attacks is also the value the price formula depends on, so its correctness is load-bearing across the whole system.

CODE QUALITY AND BEST PRACTICES REVIEW(1)

General Code Health

The Liquid Liberty Phase 1 codebase is clean, readable, and clearly written by someone who understands the problem domain. The on-chain logic is small and direct, with no inheritance gymnastics or indirection that would obscure the flow of funds. The off-chain service is organized into clear domains (signatures, shared services, handlers) with sensible separation between signing, validation, and response building.

Observations:

- The vault reads top to bottom in a logical order: immutables and state, admin functions, treasury paths, user-facing mint and burn, internal math, then views. This makes the accounting easy to follow.
- Function visibility is appropriate throughout. The share token's mint and burn are correctly restricted to the vault via onlyOwner, and nothing on the vault is more exposed than it needs to be.
- Naming is consistent and self-explanatory, both in the contracts (totalCollateral, vaultSpread, trustedSigner) and in the service code.
- Events are defined and emitted on the meaningful state changes (mint, burn, treasury set, spread set, renounce).
- The documentation shipped with the repo is well above what we usually see at this stage. The AUDIT.md, known-issues index, and inline NatSpec pre-disclose design choices and tradeoffs, which made the intent behind each decision easy to verify against the code.

Overall, readability is high and the separation between on-chain custody and off-chain authorization keeps the attack surface small.

Solidity Best Practices

Practice	Observation	Status
Pragma pinning	Compiler fixed at 0.8.20 across all contracts.	✅ Good
Safe arithmetic	Relies on 0.8.x checked math throughout. No unchecked blocks in the value-moving paths.	✅ Good
Constants and immutables	underlyingToken, vaultToken, and trustedSigner are immutable; spread tiers are named constants. Good use of immutability to close attack paths.	✅ Good
SafeERC20	All underlying-token transfers go through SafeERC20.	✅ Good
Reentrancy protection	nonReentrant applied to the user-facing mint and burn. Not applied to the owner and treasury paths.	⚠️ Partial
Signature handling	EIP-712 via OpenZeppelin, with per-user nonces, deadlines, and distinct typehashes for mint and burn.	✅ Good
Access control	Owner setters and the treasury paths are correctly gated; share-token mint and burn are vault-only.	✅ Good

CODE QUALITY AND BEST PRACTICES REVIEW(2)

Off-Chain Service Quality

The signer service is small and disciplined, and the security-relevant pieces are concentrated where they should be.

Strengths:

- The auth gate uses a length pre-check followed by crypto.timingSafeEqual, which is the correct pattern for comparing secrets, and it fails closed: when the key is unset the service refuses to operate rather than serving ungated.
- Auth failures return a flat, generic 401 with no per-reason leak, so an attacker gets no oracle for which check their token tripped.
- The mint and burn handlers reject a missing nonce explicitly rather than silently defaulting it, with a comment explaining why the on-chain nonce must be fetched. This is a thoughtful guard against a real footgun.
- The EIP-712 domain and types in the service match the contract exactly (name LibertyVault, version 3, and identical field ordering), so signatures verify on-chain as intended.

Areas to tighten:

- The signing-key discovery falls back to a well-known public Hardhat key when neither the environment variable nor the deployment-output file is present. The service logs a warning but still returns the key. This is disclosed by the project, but in a production build it should hard-fail instead of falling back.
- Both signature handlers echo the caught error.message back to the client as details. The realistic leak surface is low, but logging server-side and returning a generic message is cleaner.
- The signer handlers validate that amount parses as an integer but do not bound it (for example, rejecting zero or absurd values) before signing. On-chain checks contain the impact, so this is a robustness note rather than a vulnerability.
- The proxy functions and the core API both serve Access-Control-Allow-Origin: *. On the core API this is mitigated by the Bearer requirement, but combined with an unauthenticated proxy it is part of the M-02 picture covered in the findings.

Gas and Efficiency

Gas was not a focus of this engagement, and the contracts are already economical given their size. The use of immutables for the token and signer addresses avoids repeated storage reads, the math paths are short with no loops, and there are no unbounded iterations anywhere in the vault. No gas changes are recommended that would trade away clarity or safety.

Upgradeability and Extensibility

The contracts are non-proxy, immutable deployments by design. There is no upgrade path, no storage-layout proxy pattern, and no admin recovery or pause surface, which the project states is a deliberate choice favoring exploit resistance over operational flexibility. The consequence, which the team acknowledges, is that recovery from a signer-key compromise is operational only (the service stops signing) and there is no on-chain remediation short of leaving the vault unused. This is a reasonable position for the stated threat model as long as key custody is handled carefully.

CODE QUALITY AND BEST PRACTICES REVIEW(3)

Documentation and Comments

Documentation quality is high. The contracts carry NatSpec on functions and state variables, the math sections explain the spread-on-top and spread-from-within models, and the repo ships a dedicated audit packet, a known-issues index, and provenance notes. This level of disclosure is uncommon at handoff and made design intent easy to check against the code. One caveat: several key safety properties are documented as prose claims (for example, price monotonicity). We treated those as claims to verify rather than facts, and one holds only with a boundary exception, captured in the findings.

Testing Coverage

The in-repo Hardhat suite has 122 passing tests with good coverage of documented behavior: mint and burn happy paths, spread math at both tiers, signature expiry and nonce replay, the renounce and bootstrap flags, and reentrancy via a malicious underlying mock. The gap is in adversarial and boundary coverage, which is where SpyWolf focused its own simulation work: the supply-to-zero boundary, explicit value conservation, share-inflation attempts, and round-trip profitability.

Overall Code Quality Assessment

Code quality is strong. The structure is clean, the use of audited libraries is correct, the documentation is thorough, and the design deliberately closes several common vault attack classes. Our recommendations are mostly procedural and defensive: close the price-formula boundary at supply zero, strengthen the off-chain authorization so the auth gate cannot be bypassed through the public proxy, harden the production signer config so it cannot fall back to a public key, extend nonReentrant to the remaining paths, and pin the EVM target for the deployment chain. With these addressed, the in-scope surface reaches a high standard of correctness and maintainability for its intended Phase 1 deployment.

FINDINGS AND VULNERABILITY ANALYSIS(1)

Methodology for Severity Rating

Each issue was assigned a severity level based on two dimensions:

Severity	Impact	Likelihood	Description
Critical	Direct loss of funds or total system compromise	High	Must be fixed before deployment
High	Potential loss of funds or systemic malfunction	Medium–High	Priority for immediate remediation
Medium	Logic or validation flaws that can cause local issues	Medium	Should be fixed before mainnet
Low	Minor logic errors, inefficiencies, or incomplete patterns	Low	Recommended improvement
Informational	Cosmetic or stylistic issues	—	Optional adjustments

Severity was determined by **impact × exploitability**, emphasizing how realistic an attacker scenario is under the given system assumptions (trusted team governance, EVM equivalence on PulseChain, etc.).

Severity reflects impact multiplied by exploitability. A finding that breaks a documented invariant but is hard to reach in the live configuration, or whose worst case is commercial rather than user-fund loss, is rated Medium rather than High. Where a finding's severity would change under a different deployment assumption, that escalation condition is stated in the finding itself.

FINDINGS AND VULNERABILITY ANALYSIS(1)

Findings Summary

ID	Title	Severity	Status
F-01	Full-redemption price reset and orphaned collateral at supply zero	🟨 Medium	✅ FIXED
F-02	Signature auth gate bypassable through the public proxy	🟨 Medium	✅ FIXED
F-03	Production signer can fall back to a public Hardhat key	Low	✅ FIXED
F-04	Non-standard underlying breaks collateral accounting	Low	✅ FIXED
F-05	EVM version not pinned for the PulseChain target	Low	✅ FIXED
F-06	nonReentrant absent on owner and treasury paths	Informational	✅ FIXED
F-07	Signer handlers do not bound amount; error messages echoed	Informational	✅ FIXED
F-08	renounceAdminControl does not override inherited Ownable functions	Informational	✅ FIXED

DETAILED FINDINGS

~~F-01: Full-Redemption Price Reset and Orphaned Collateral at Supply Zero (1)~~

Component: Core / LibertyVault.sol

Description:

The vault prices its token as totalCollateral / supply and treats an empty supply as a bootstrap condition. When the entire supply is burned to zero, two things happen together.

First, getCurrentPrice() (and getVaultTokenPrice()) stop returning the accrued price and fall back to a hardcoded 1e18 sentinel because of the currentSupply == 0 branch. A vault that had grown its price well above 1.0 reports 1.0 again the instant supply hits zero. This breaks the protocol's first stated invariant in PHASED_LAUNCH_PLAN §7.2, that getCurrentPrice() is monotonically non-decreasing.

Second, the spread retained on that final burn stays in totalCollateral while supply is zero, backing no tokens. The next mint() re-enters the 1:1 bootstrap branch in _calculateMintAmount (the currentSupply == 0 || totalCollateral == 0 condition) and prices new shares at 1.0 even though the vault already holds that orphaned collateral. The first minter after the reset acquires it for free and can withdraw it immediately with a burn.

Impact:

A stated protocol invariant is violated, and collateral retained from the final burn detaches from the holders who funded it and becomes capturable. This is not a drain of active deposits, since value is otherwise conserved, but it is a leak of accrued backing and a break in a property the system documents as always holding. Severity rises to High if getCurrentPrice() is ever consumed on-chain as an oracle by a future peer contract, since a forced reset to 1.0 would become a price-manipulation primitive. The function is described in the code as existing "for oracle compatibility," which makes this a realistic path.

Likelihood: Low to Medium

Reaching supply zero is not possible in the intended Phase 1 topology, because the seeded PulseX V2 pool permanently holds vault tokens and a constant-product pool cannot have its reserve fully bought out. It is reachable before liquidity is seeded, in any deployment run without a permanent pool, and in the "config redeploy with a different underlying" scenario the project describes.

Affected Lines: _calculateMintAmount bootstrap condition (~line 329); getCurrentPrice and getVaultTokenPrice zero-supply branches (~lines 367 to 385); burn collateral reduction (~lines 297 to 309).

DETAILED FINDINGS

~~F-01: Full-Redemption Price Reset and Orphaned Collateral at Supply Zero (2)~~

Simulation Steps:

1. Bootstrap the vault and drive the price above 1.0 through ordinary spread retention.
2. Burn the entire supply to zero in a single redemption.
3. Observe getCurrentPrice() reset to 1e18 while totalCollateral stays positive.
4. Mint a minimal amount (priced 1:1 against the empty vault), then burn it, and confirm the caller withdraws the orphaned collateral at a profit.

In our exact-integer model, a vault holding roughly 1.22M units of collateral left about 12,200 orphaned after full redemption (approximately the active spread on the pre-redemption collateral). A one-token mint then burn returned about 12,095 to the attacker against an outlay near 1.01. The orphaned amount scales linearly with vault size at the moment of full redemption.

Status: Fixed

Category: Accounting Consistency / Invariant Integrity

Implemented Fixes:

Resolved in v0.1.1-audit-patched. Bootstrap now mints 1,000 wei of dead shares to 0xdead and the mint bootstrap branch is gated on currentSupply == 0 only. Supply can no longer reach zero, so the price never resets and no collateral is orphaned. Verified by re-running our model against the patched logic: monotonicity holds across full redemption, with no division-by-zero and a negligible 1e-15 initial price drift.

DETAILED FINDINGS

~~F-02: Signature Auth Gate Bypassable Through the Public Proxy (1)~~

Component: Off-chain / apps/web/netlify/functions/, apps/core-api/src/shared/services/auth.service.ts

Description:

Every mint and burn requires an EIP-712 signature from the trustedSigner. To stop arbitrary parties from obtaining those signatures and running their own bot on the operator's infrastructure, the signer endpoint is gated by a shared SIGNER_API_KEY Bearer token. The documented intent in AUDIT.md §5.6 is that "only our UI and bots can request signatures."

The auth gate itself is implemented correctly: it uses a length pre-check followed by crypto.timingSafeEqual, returns a generic 401 with no per-failure leak, and fails closed when the key is unset. The problem is upstream of it. The browser-facing proxy functions (request-mint-signature.ts and request-burn-signature.ts) inject the SIGNER_API_KEY for any caller and serve Access-Control-Allow-Origin: *. The proxy URL is a public Netlify endpoint, trivially discoverable in the site's network traffic or bundle. Anyone who reaches it can POST a request and receive a valid signature, because the proxy adds the secret on their behalf.

The net effect is that the auth gate restricts direct calls to the core API but not calls through the public proxy, which is the path everyone can reach. The control does not achieve its stated objective.

Impact:

The commercial gate the signer layer is meant to enforce is effectively open. Any party can obtain mint and burn signatures and run their own arbitrage against the vault, diluting the arb-revenue monopoly that funds later phases. The endpoint is also unrate-limited, so the open proxy is a free signing and resource-exhaustion surface against the operator's infrastructure.

This is commercial impact, not protocol fund loss. Because the contract binds each signature to msg.sender, a party obtaining signatures can only act on their own balance and still pays the full spread. They cannot mint or burn on behalf of another user, drain the vault, or bypass the on-chain signature check.

Likelihood: High

No special access or key knowledge is required. The proxy laundering the secret for arbitrary callers is the default behavior of the shipped code.

Affected Lines: request-mint-signature.ts and request-burn-signature.ts (API-key injection and CORS headers); DEFAULT_CORS_CONFIG in the shared HTTP constants.

DETAILED FINDINGS

Medium Risk

✅ FIXED

~~F-02: Signature Auth Gate Bypassable Through the Public Proxy (2)~~

Simulation Steps:

1. From an unrelated origin, POST a well-formed mint-signature request to the public proxy URL with no API key.
2. Confirm the proxy injects the Bearer token, forwards to the core API, and returns a valid signature.
3. Submit the signature on-chain from the requesting address and confirm the mint succeeds.

Status: Fixed

Category: Access Control / Authorization

Implemented Fixes:

Resolved in v0.1.1-audit-patched. Both proxies now route through a single gate: origin allowlist (fail-closed), per-wallet and per-IP rate limiting, and responses that echo the validated origin instead of *. Core-api CORS was also moved off *. This raises abuse cost to the level the finding required; the on-chain msg.sender binding remains the hard guarantee against fund loss.

DETAILED FINDINGS

~~F-03: Production Signer Can Fall Back to a Public Hardhat Key~~

Component: Off-chain / apps/core-api/src/shared/services/signer-config.service.ts

Description:
getSignerPrivateKey() resolves the signing key in three steps: the SIGNER_PRIVATE_KEY environment variable, then a deployment-output/signer-config.json file, and finally a hardcoded fallback to the well-known Hardhat account #1 private key. The fallback logs a "NOT FOR PRODUCTION" warning but still returns the key, so the service keeps signing rather than refusing to start.

If a production deploy is misconfigured and neither the environment variable nor the config file is present, the service signs with a private key that is public and known to everyone. At that point any party can forge valid trustedSigner signatures.

Impact:
The signature gate is fully defeated if the fallback is ever reached in production. The blast radius is still bounded by the on-chain msg.sender binding, so a forger can only mint and burn on their own balance and still pays the spread; they cannot drain the vault or act for other users. The impact is therefore commercial rather than direct fund loss, the same class as F-02, but the likelihood of misconfiguration makes it worth closing before launch. The project discloses this in AUDIT.md §6.3.

Likelihood: Low

Requires an operator misconfiguration at deploy time. Real, but within the team's control.

Affected Lines: getSignerPrivateKey() and the HARDHAT_DEFAULT_SIGNER_KEY constant in signer-config.service.ts.

Status: Fixed

Category: Key Management / Configuration Safety

Implemented Fixes:

Resolved in v0.1.1-audit-patched. The Hardhat key fallback is now gated behind ALLOW_HARDHAT_FALLBACK === "true" and hard-fails with a config error otherwise, and isSignerConfigured() reflects real-key presence only. A misconfigured production deploy now refuses to sign rather than signing with a public key.

DETAILED FINDINGS

~~F-04: Non-Standard Underlying Breaks Collateral Accounting~~

Component: Core / LibertyVault.sol

Description:
The vault credits totalCollateral by the amount it intends to transfer rather than by the actual balance received. In mint, it does safeTransferFrom(user, vault, totalRequired) and then totalCollateral += totalRequired. bootstrapLiquidity follows the same pattern. This is correct for a standard ERC-20, but it assumes the amount sent equals the amount received.

If the underlying is a fee-on-transfer, rebasing, or hook-bearing token, that assumption fails. A transfer-fee token would deliver less than totalRequired while the vault still credits the full figure, so totalCollateral would overstate the real balance. Over time the vault becomes under-collateralized and the last redeemers cannot be fully paid.

Impact:
Under a non-standard underlying, accounted collateral drifts above the real balance, eventually leaving the vault unable to honor all redemptions. The collateral token is immutable and operator-chosen, and Phase 1 uses bridged DAI, which is standard, so this is not exploitable in the intended deployment. It is recorded as a Low because bridged DAI inherits the behavior of its bridge, and any future redeploy with a different underlying would reopen the risk.

Likelihood: Low

Requires deploying against a fee-on-transfer or rebasing underlying, which the current configuration does not do.

Affected Lines: mint collateral credit (~lines 266 to 270); bootstrapLiquidity collateral credit (~lines 159 to 163); treasuryDeposit collateral credit (~lines 194 to 198).

Status: Fixed

Category: Accounting Consistency / Token Compatibility

Implemented Fixes:

Resolved in v0.1.1-audit-patched. bootstrapLiquidity, treasuryDeposit, and mint now credit totalCollateral by the measured balance delta rather than the requested amount, so accounting stays correct even if a transfer delivers less than requested.

DETAILED FINDINGS

~~F-05: EVM Version Not Pinned for the Deployment Target~~

Component: Build configuration / hardhat.config.js

Description:
hardhat.config.js sets the compiler to 0.8.20 with the optimizer enabled but does not set an evmVersion. With no explicit target, the build uses the compiler default for 0.8.20, which is Shanghai, so the generated bytecode includes the PUSH0 opcode. PUSH0 (EIP-3855) only executes on chains that have adopted the Shanghai upgrade; on chains that have not, the opcode is treated as invalid and deployment or execution fails.

This is a correctness-and-determinism concern rather than a contract logic flaw. Leaving the target implicit means the bytecode that ships depends on a compiler default rather than a deliberate choice matched to the deployment chain.

Impact:
If the contracts were ever deployed to a chain that does not support PUSH0, deployment would fail outright. For the current PulseChain target this appears to be a non-issue, since the project already compiles and deploys the same 0.8.20 contracts on PulseChain testnet during its drills, which indicates the chain supports the opcode. The practical impact is therefore limited to build reproducibility and to the risk surface of any future redeploy on a different chain. Pinning the target removes the dependency on a compiler default and makes the intended environment explicit.

Likelihood: Low

No effect on the current deployment path. The risk only materializes on an unverified or different target.

Affected Lines: solidity.settings block in hardhat.config.js (no evmVersion key).

Status: Fixed

Category: Build Configuration / Determinism

Implemented Fixes:

Resolved in v0.1.1-audit-patched. evmVersion: "paris" is now pinned in hardhat.config.js, making the shipped bytecode a deliberate, PUSH0-free choice rather than a compiler default.

FINDINGS

Informational

~~F-06: nonReentrant Absent on Owner and Treasury Paths~~ ✅ FIXED

Component: Core / LibertyVault.sol

The user-facing mint and burn are guarded by nonReentrant, but bootstrapLiquidity, treasuryDeposit, and treasuryWithdraw are not. These paths are restricted to the owner or the treasury and rely on an immutable, trusted underlying token, so there is no reachable reentrancy in Phase 1, and the project discloses this in AUDIT.md §6.2. We agree it is low risk as deployed, but recommend extending nonReentrant to all three as defense-in-depth when the Treasury is wired in a later phase.

Implemented Fixes: nonReentrant added to bootstrapLiquidity, treasuryDeposit, treasuryWithdraw

~~F-07: Signer Handlers Do Not Bound amount; Error Messages Echoed~~ ✅ FIXED

Component: Off-chain / vault-mint-signature.handler.ts, vault-burn-signature.handler.ts

The signature handlers confirm that amount parses as an integer but do not reject zero or otherwise unreasonable values before signing. A zero produces a signature that simply reverts on-chain, so the on-chain checks contain the impact, but validating amount > 0 server-side is cleaner. Separately, both handlers return the caught error.message to the client as details; the realistic leak surface is low, but logging the message server-side and returning a generic response is the better pattern. Both are hygiene items with no exploit path.

Implemented Fixes: amount > 0 validated server-side (returns 400); error.message no longer echoed, logged server-side instead

~~F-08: renounceAdminControl Does Not Override Inherited Ownable Functions~~ ✅ FIXED

Component: Core / LibertyVault.sol

After renounceAdminControl(), the notRenounced modifier locks every protocol-specific owner setter, but the inherited Ownable.transferOwnership and Ownable.renounceOwnership remain callable. The practical effect is null, since every state-changing setter is already gated by notRenounced, so a transferred or renounced ownership inherits no usable power. The project discloses this in AUDIT.md §6.1. We recommend overriding both inherited functions to revert once renounced, purely so the contract's surface matches its intent.

Implemented Fixes: transferOwnership / renounceOwnership overridden to require(!renounced); pre-renounce owner→multisig migration preserved

SIMULATION AND TESTING SUMMARY(1)

Overview

To confirm the findings and validate the vault's economic properties, SpyWolf reviewed the project's in-repo Hardhat suite and then built an independent model of the vault's arithmetic that reproduces Solidity 0.8.20 integer behavior, including floor division and checked over- and underflow. The model was exercised under conditions the happy-path tests do not reach, with the emphasis on the economic invariants the protocol documents as always holding.

The testing combined:

- Review of the in-repo Hardhat suite (122 passing tests) for coverage and correctness of its assertions.
- Property and invariant checks over several hundred randomized mint and burn sequences with deterministic seeds for reproducibility.
- Targeted boundary tests around empty supply, first deposit, and minimal amounts.
- Adversarial sequences aimed at value extraction, share inflation, and signature reuse.

All work used the same compiler configuration the project ships (solc 0.8.20).

Key Simulation Scenarios and Results

ID	Scenario	Objective	Expected Behavior	Outcome
S-01	Price monotonicity	Confirm getCurrentPrice never decreases across mixed sequences	Non-decreasing at every step	Holds, except supply-to-zero
S-02	Supply-to-zero boundary	Burn entire supply, then re-mint	Price preserved across redemption	Invariant break confirmed (F-01)
S-03	Vault solvency	Held collateral vs accounted collateral	Held >= accounted at all times	Passed
S-04	Value conservation	Sum of deposits vs withdrawals plus backing	Conserved to the wei	Passed
S-05	Share inflation / donation	First-depositor and direct-donation attempts	No share-price manipulation	Passed (internal accounting)
S-06	Round-trip profitability	Mint then immediate burn	Caller never profits	Passed (always net loss)
S-07	Signature replay and reuse	Reuse, nonce reuse, mint/burn cross-type	Revert on reuse or wrong type	Passed
S-08	Reentrancy via underlying	Reentrant call on mint and burn	Blocked by nonReentrant	Passed
S-09	Auth gate branches	Missing, malformed, wrong-length, correct key	Reject all but correct key	Passed

SIMULATION AND TESTING SUMMARY(2)

Invariant and Property Testing

Invariant	Description	Result
Price Monotonicity	getCurrentPrice non-decreasing across any sequence	Holds, with the supply-to-zero exception in F-01
Over-Collateralization	Outstanding supply at current price never exceeds collateral	Holds true
Liquidity Conservation	Deposits equal withdrawals plus collateral still backing supply	Holds true
No Value Extraction	No attacker-controlled actor withdraws more than deposited	Holds true
Nonce Integrity	Per-user nonce strictly increments; no replay possible	Holds true
No Underflow/Overflow	Arithmetic safe across mint, burn, and bootstrap	Holds true

Test Environment

Parameter	Value
Compiler	Solidity 0.8.20
Optimizer	Enabled, 200 runs
In-Repo Suite	Hardhat, 122 passing
Simulation	Independent exact-integer model, deterministic seeds
Sequences	Several hundred randomized mint/burn runs
RPC Fork	Not required; all simulations performed locally

Conclusion of Testing

The simulations confirmed both the strengths and the one weakness identified in the review. The vault remained solvent and value-conservative across every randomized sequence, no attacker-controlled actor was able to extract more than it deposited, and the standard inflation and round-trip attacks were ruled out by the internal accounting model. The only property that did not hold universally was price monotonicity, which breaks at the supply-to-zero boundary documented in F-01. These property checks can be carried forward as a regression suite once the findings are remediated.

INTEGRATION & ECONOMIC SAFETY REVIEW

Objective

Confirm that the on-chain vault and the off-chain signer and proxy layers interact safely, that collateral accounting stays consistent across mint, burn, and bootstrap, and that the vault's economic design behaves as intended under sustained activity. Because Phase 1 has no peer contracts, plugins, or automation vaults, the integration surface is narrow: a single vault, its share token, and the signing path that authorizes every operation.

Trust Boundary and Interaction Review

The central boundary is between the on-chain custody layer and the off-chain authorization layer. The contract can prove a signature came from the trustedSigner and enforce per-user nonces and deadlines, but it cannot know who requested that signature off-chain. This split is by design and is the reason the auth gate is a commercial control rather than a protocol-safety control. The on-chain msg.sender binding is what actually protects users: a signature is only usable by the address it names, paying from and receiving to that same address.

Interaction	Key Concern	Result
Vault <-> Share Token	Mint/burn access control	Safe; share token mint and burn are vault-only
Vault <-> Underlying	Collateral accounting on transfer	Correct for a standard token; see F-04 for non-standard underlying
User <-> Signer/Proxy	Who can obtain a signature	Auth gate bypassable via public proxy; see F-02
Contract <-> Signature	On-chain signer and nonce check	Safe; bound to msg.sender, nonce strictly increments
Caller <-> Mint/Burn	Round-trip and inflation safety	Safe; conservative and non-extractable except at supply zero (F-01)

Economic Invariants

The vault's value accrual was checked against the properties the protocol documents:

- **Conservation of value:** total collateral in equals total collateral out plus the collateral still backing outstanding supply, confirmed to the wei across randomized sequences.
- **Spread retention:** every mint and burn retains the active spread as permanent backing, so the price ratchets upward through ordinary activity.
- **Redemption integrity:** burning returns the proportional underlying value minus spread, and the vault stays solvent so all holders can be paid.
- **Price behavior:** price is non-decreasing across any sequence, with the single supply-to-zero exception in F-01.

All economic invariants held during simulation, with that one boundary exception and only sub-unit rounding deltas otherwise, all favoring the vault.

FINAL STATEMENT & CONCLUSION

SpyWolf performed a full technical security review of Liquid Liberty Protocol Phase 1, covering the LibertyVault and LibertyVaultToken contracts, the EIP-712 signer service, the Bearer-token auth gate, and the server-side proxy layer. The review combined line-by-line manual analysis, pattern-based static analysis, and an independent model of the vault's arithmetic exercised across several hundred randomized and adversarial sequences.

The codebase was strong from the first submission. It is clean, well-structured, and unusually well-documented, and the design deliberately closes several of the attack classes that most often compromise share-based vaults: pricing reads an internal collateral accumulator rather than the live token balance, which neutralizes the classic donation and first-depositor inflation attack; the trusted signer is immutable; bootstrap is one-shot; and every mint and burn is bound on-chain to msg.sender, keeping the blast radius of any signer or key compromise commercial rather than a threat to user funds. This is a higher engineering baseline than we typically see at audit handoff, and it made the review efficient.

The initial review identified no Critical and no High-severity issues, two Medium-severity findings, and a set of Low and Informational items. The team's response to those findings was equally strong. Every one of the eight findings (F-01 through F-08) was remediated in the patched revision v0.1.1-audit-patched, each fix accompanied by dedicated tests. The remediations are precise and proportionate, they address the root cause rather than the symptom, and they introduced no new issues. Notably, the F-01 supply-to-zero fix was verified by re-running our model against the patched logic: price monotonicity now holds unconditionally, with no division-by-zero and only a negligible sub-wei price drift at bootstrap.

We reviewed the full remediation diff and re-verified the affected logic against our own simulations. All eight findings are confirmed resolved. Under the defined trust assumptions, the in-scope contracts demonstrate strong correctness across collateral accounting, spread retention, price behavior, and signature handling, with no path to user-fund loss identified in either the original or the patched code.

Based on this review and the verified remediations, **SpyWolf considers the in-scope Liquid Liberty Phase 1 contracts ready for production deployment** under the stated trust assumptions. As standard practice, we recommend a final green run of the full test suite on the deployed tag, ongoing monitoring, careful key custody, and a follow-up review of any new components as the protocol expands into later phases.

Our thanks to the Liquid Liberty team for a well-engineered codebase, thorough documentation, and a fast, high-quality remediation cycle. It was a pleasure to work with a team that takes security this seriously.

SPYWOLF

CRYPTO SECURITY

Audits | KYCs | dApps
Contract Development

ABOUT US

We are a growing crypto security agency offering audits, KYCs and consulting services for some of the top names in the crypto industry.

- **OVER 1000 SUCCESSFUL CLIENTS**
- **MORE THAN 1000 SCAMS EXPOSED**
- **MILLIONS SAVED IN POTENTIAL FRAUD**
- **PARTNERSHIPS WITH TOP LAUNCHPADS, INFLUENCERS AND CRYPTO PROJECTS**
- **CONSTANTLY BUILDING TOOLS TO HELP INVESTORS DO BETTER RESEARCH**

To hire us, reach out to contact@spywolf.co or t.me/joe_SpyWolf

FIND US ONLINE

SPYWOLF.CO

@SPYWOLFNETWORK

@SPYWOLFNETWORK

Disclaimer

This report shows findings based on our limited project analysis, following good industry practice from the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, overall social media and website presence and team transparency details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report.
While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the disclaimer below – please make sure to read it in full.

DISCLAIMER:
By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any and all copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis, and does not constitute investment advice.
No one shall have any right to rely on the report or its contents, and SpyWolf and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers and other representatives) (SpyWolf) owe no duty of care towards you or any other person, nor does SpyWolf make any warranty or representation to any person on the accuracy or completeness of the report.
The report is provided "as is", without any conditions, warranties or other terms of any kind except as set out in this disclaimer, and SpyWolf hereby excludes all representations, warranties, conditions and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, SpyWolf hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against SpyWolf, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of use of this report, and any reliance on this report. The analysis of the security is purely based on the smart contracts, website, social media and team.
No applications were reviewed for security. No product code has been reviewed.