Smart Contract Security Audit

NoxExitVault

Target contract	0x3cfe2fcf1724516455E52D4e3624F3ef44D75218
Network	Ethereum Mainnet
Contract name	NoxExitVault
Compiler	solc v0.8.24+commit.e11b9ed9 (optimizer: 200 runs)
EVM version	cancun
License	MIT
Source verification	Verified on Etherscan (exact match)
Report date	July 15, 2026
Audit type	Manual source-code security review

This report reflects a point-in-time review of the source code listed above. It is not a guarantee of security and does not constitute investment advice. See the full disclaimer in Appendix B.

1. Executive Summary

NoxExitVault is an Ethereum mainnet liquidity-provider (LP) vault that pre-fills “fast exits” from an L2 (“DBK”). LPs deposit ETH and receive pro-rata shares. A designated off-chain **relayer** observes `FastExitRequested` events on the L2 and calls `fillExit` to pay the exiting user 90% of their gross exit amount immediately from vault liquidity; the vault retains a 10% spread (split between a relayer gas refund, an admin fee, and LP yield). An operator later recycles the ETH through the canonical 7-day bridge and returns it via `repay()` or a plain transfer.

The contract is short (273 lines), dependency-free, and generally well written: it uses Solidity 0.8.24 (built-in overflow checks), custom errors, access-control modifiers, events on all state changes, and correct checks-effects-interactions ordering. No vulnerability was identified that allows an *unprivileged external attacker* to steal funds.

The dominant risks are **trust and centralization**: the relayer can direct LP funds to arbitrary recipients with no on-chain proof that a real L2 exit exists, and the owner can lift the caps that bound that damage. Share-price accounting also excludes in-transit funds, which lets sophisticated depositors extract value from passive LPs.

Severity	Count	Finding IDs
High	2	H-01, H-02
Medium	2	M-01, M-02
Low	5	L-01 – L-05
Informational	2	I-01, I-02

2. Scope and Methodology

In scope. The single verified contract `NoxExitVault` deployed at `0x3cfe2fcf1724516455E52D4e3624F3ef44D75218` on Ethereum mainnet (273 lines of Solidity, no external imports). Constructor arguments: relayer `0xb6D7a3Ec61c5D0276fc458b2a71ABb3b279f6B7e`, treasury `0xe86ab361A1830C67ecE91d2eB935965Ff50451c5`.

Out of scope. The DBK L2 contracts and bridge, the off-chain relayer and operator infrastructure, key management, and the treasury contract.

Methodology. Line-by-line manual review of the verified source against a checklist covering: reentrancy, integer overflow/underflow, unchecked external calls, access control, low-level call usage, external calls in loops, gas-limit/DoS vectors, randomness dependence, denial-of-service, front-running/MEV, share-accounting attacks (first-depositor inflation, donation attacks), timestamp dependence, and code complexity/maintainability. The deployed bytecode is an Etherscan “exact match” for the reviewed source.

3. Contract Overview

Roles. *Owner* (deployer): sets parameters, pauses, sweeps fees, rotates the relayer/treasury, transfers ownership. *Relayer*: the only address allowed to call `fillExit`. *Treasury*: receives swept admin fees. *LPs*: deposit and withdraw ETH via a share system.

Flows. (1) LP deposits ETH → shares minted at the current share price with a virtual-offset formula `shares = amount × (totalShares + 1000) / (preAssets + 1)`; a per-address withdrawal lock (default 1 day) starts/renews on every deposit. (2) Relayer calls `fillExit(id, recipient, gross)`; the contract pays `net = 90% × gross` to the recipient, refunds up to 1% (capped at `maxRefund`) to the relayer, accrues the remainder of that 1% as admin fees, and implicitly leaves ~9% in the vault as LP spread. Fills are bounded

per-transaction (`maxGross`, default 0.5 ETH) and per-UTC-day (`dailyCap`, default 10 ETH), and replay-protected by the `filled[id]` mapping. (3) The operator repays recycled ETH via `repay()` or a plain transfer to `receive()`, which decrements the informational `inTransit` counter.

Accounting. `availableLiquidity() = balance − fees`; `totalAssets()` equals available liquidity, i.e. funds currently out on fills (`inTransit`) are *not* counted as LP assets (see M-01).

4. Detailed Findings

H-01 — Relayer can pay arbitrary recipients with no proof of a real L2 exit — High

Location: `fillExit(), lines 158–188`

Description. fillExit accepts an arbitrary `id`, `recipient`, and `gross` from the relayer with no on-chain verification that a corresponding `FastExitRequested` event actually exists on the L2, that the recipient matches the L2 exiter, or that the amount is correct. The entire safety of LP funds rests on the honesty and operational security of one externally-owned relayer key.

Impact. A compromised or malicious relayer can invent exit IDs and drain `dailyCap` (10 ETH by default) of LP funds per UTC day — up to ~2× that in minutes by straddling a UTC-midnight boundary (see M-02) — to any address, until the owner notices and pauses or rotates the relayer.

Attack scenario. Attacker obtains the relayer private key, then repeatedly calls `fillExit(keccak256(i), attacker, 0.5 ether)` (20 calls per day at default parameters). Each call transfers 0.45 ETH to the attacker plus a 0.005 ETH self-refund; unique fabricated IDs bypass replay protection.

Recommendation. Remove the single-key trust assumption where feasible. Verify each exit using a canonical bridge proof, light-client/state-root proof, or a threshold-signed attestation whose signers are independent. At minimum, use a multisig/threshold relayer, strict immutable hard caps, real-time monitoring, automated pausing, and isolated keys with rotation and incident-response procedures. Document this trust assumption prominently for LPs.

H-02 — Owner can remove the limits that bound relayer losses — High

Location: `setDailyCap(), lines 233–236; setMaxGross(), lines 238–241`

Description. The owner may set `dailyCap` and `maxGross` to any uint256 value with immediate effect. This is inconsistent with `maxRefund` and `withdrawLock`, which have hard upper bounds. The owner cannot directly call `fillExit`, but it can remove the controls that limit a compromised relayer, or appoint itself/another controlled address as relayer.

Impact. Compromise or misuse of the owner key can turn the bounded H-01 loss into a drain of all available LP liquidity in one transaction or block. Because changes are immediate, LPs have no guaranteed window in which to exit after detecting a dangerous change.

Attack scenario. An attacker compromises the owner, sets both caps to the vault balance, appoints an attacker-controlled relayer, and calls `fillExit` with a fake ID and attacker recipient.

Recommendation. Enforce immutable protocol-level ceilings for both parameters. Put the owner behind a multisig and route sensitive changes through a timelock long enough for LPs and monitors to react. Consider separate roles for pausing (fast emergency action) and delayed governance (cap increases and role changes). Cap decreases may remain immediate.

M-01 — In-transit receivables are excluded from share value — Medium

Location: `availableLiquidity()/totalAssets(), lines 95–103; fillExit(), lines 176–182`

Description. `totalAssets()` counts only the vault's current ETH balance minus fees. It ignores `inTransit`, even though an exit fill creates an expected bridge receivable. A fill therefore reduces the apparent share price immediately, while a later repayment increases it. New shares minted during this low-NAV period receive a claim on repayments attributable to pre-existing LP capital.

Impact. A depositor who times entry after fills and withdrawal after repayment can capture part of the receivable and fee spread without bearing the original liquidity exposure, diluting passive LPs. Public mempool visibility and the predictable bridge delay make this timing strategy practical. The exact gain depends on fill volume, outstanding shares, and when repayment occurs.

Attack scenario. Suppose the vault has 100 ETH and 100-equivalent shares. A 10 ETH gross fill pays roughly 9.1 ETH, reducing accounted assets to 90.9 ETH while recording 10 ETH in transit. An attacker deposits at the depressed price, waits for the 10 ETH repayment and withdrawal lock, then exits with a share of value generated by the earlier LP-funded fill.

Recommendation. Define a consistent NAV model. If bridge repayments are enforceable/creditworthy, include a conservatively valued in-transit receivable in `totalAssets()` (and separately account for expected spread). Otherwise, prevent or queue deposits and withdrawals while receivables are outstanding, or use epoch-based settlement so all LPs enter and exit at a finalized NAV. Add invariant and adversarial timing tests.

M-02 — Calendar-day cap permits nearly twice the cap around midnight — Medium

Location: `dailyRemaining(), lines 111–115; fillExit(), lines 167–180`

Description. The rate limit is keyed by `block.timestamp / 1 days`. This is a calendar-day bucket, not a rolling 24-hour window. A relayer may consume the entire cap immediately before UTC midnight and consume a fresh cap immediately after midnight.

Impact. Up to almost 2× `dailyCap` can be filled within minutes or even adjacent blocks, weakening the intended loss and liquidity-rate bound. This compounds H-01 during relayer compromise.

Attack scenario. At 23:59:59 UTC, the relayer fills 10 ETH. At 00:00:01 UTC, it fills another 10 ETH. Both batches pass despite 20 ETH leaving the system within seconds.

Recommendation. Use a rolling-window limiter, such as a fixed-size ring buffer of hourly buckets summed over 24 hours, or a token-bucket/leaky-bucket algorithm with bounded replenishment. If calendar buckets are intentional, document that the effective burst limit is 2× and set the configured cap accordingly.

L-01 — External ETH transfers lack an explicit reentrancy guard — Low

Location: `_send(), lines 268–271; withdraw(), fillExit(), sweepFees()`

Description. `_send` uses a low-level call that forwards all remaining gas. The current state-changing callers follow checks-effects-interactions: withdrawal shares are burned before sending, fill/replay/cap/accounting state is committed before both sends, and fees are zeroed before sweeping. Consequently, no exploitable reentrancy path was identified in the reviewed version. However, callbacks can re-enter other public methods and future changes could invalidate this implicit safety argument.

Impact. The present risk is defense-in-depth and maintenance risk rather than a demonstrated fund loss. A malicious recipient can also consume gas or revert, causing the operation to fail.

Recommendation. Add a standard reentrancy guard to external functions that transfer ETH, and retain CEI. Consider a pull-payment model where practical. Add receiver contracts to the test suite that re-enter each externally callable function and that revert or consume gas.

L-02 — Plain ETH transfers are treated as repayments and can silently lose user funds — Low

Location: `receive(), lines 195–197; _repay(), lines 199–207`

Description. Any plain ETH transfer calls `_repay`, reduces `inTransit`, and mints no shares. The sender is not authenticated as the bridge/operator and there is no way to distinguish a repayment from a donation or mistaken transfer. Furthermore, `inTransit` does not constrain fills, withdrawals, or share valuation; it is currently informational accounting only.

Impact. Users or integrations that send ETH directly instead of calling `deposit()` permanently donate it to LPs. Third parties can also make operational telemetry inaccurate by reducing `inTransit` without a corresponding canonical bridge repayment.

Recommendation. Prefer an explicit authenticated repayment function carrying a bridge/exit identifier, with reconciliation against outstanding receivables. Make `receive` revert, or clearly document that all direct transfers are irrevocable donations and emit a distinct donation event. If open repayment is required, do not treat its counter as an authoritative operational invariant.

L-03 — A small top-up re-locks all of a user's shares — Low

Location: `deposit(), lines 130–134; withdraw(), line 141`

Description. Every deposit replaces `withdrawUnlock[msg.sender]` with the current time plus `withdrawLock`. Since shares are pooled per address, even a minimum-size top-up re-locks the user's previously matured position.

Impact. This creates surprising UX and may block a planned withdrawal for up to seven days if the owner has raised the lock. The user causes the top-up themselves, so this is not directly griefable by an unrelated address.

Recommendation. Track locked share tranches independently, or preserve the unlock of existing shares and apply the new lock only to newly minted shares. If the behavior is intentional, make it explicit in interfaces and documentation and require a clear user confirmation.

L-04 — Ownership transfer is single-step and emits no event — Low

Location: `transferOwnership(), lines 262–265`

Description. Ownership is transferred immediately to the supplied nonzero address, with no acceptance step and no `OwnershipTransferred` event. A typo or incompatible recipient can permanently strand privileged operations, while monitoring systems cannot reliably discover the change from a dedicated event.

Impact. Loss of owner control can prevent emergency pausing and role/cap management. Transfer to an unintended but controlled address can also create an unexpected privileged party.

Recommendation. Use a two-step transfer pattern such as OpenZeppelin `Ownable2Step`, where the pending owner must accept. Emit events for initiation and acceptance. Use a multisig as the production owner.

L-05 — Reverting recipients can block individual fast exits — Low

Location: `fillExit(), lines 184–185; _send(), lines 268–271`

Description. Fast exits push ETH directly to the recipient. If the recipient is a contract that rejects ETH or deliberately reverts, the whole fill reverts, including the `filled[id]` state update. The relayer refund transfer can similarly fail if the relayer is a contract without a payable receiver.

Impact. The affected exit cannot be completed to that recipient through this function. No state or funds are lost because the transaction reverts atomically, but service availability is reduced and relayer contract upgrades can unexpectedly stop all fills.

Recommendation. Allow users to specify or update a payable L1 recipient in the originating request. Alternatively, credit claimable balances and let recipients pull ETH. Validate relayer contract receiver behavior

before role assignment.

I-01 — Virtual offset mitigates but does not eliminate donation-driven pricing effects — Informational

Location: `deposit(), lines 122–128; receive(), lines 195–197`

Description. The share formula uses virtual assets/shares `(totalShares + 1000) / (preAssets + 1)`, which substantially dampens the classic first-depositor inflation attack. The minimum deposit and withdrawal lock further increase attack cost. Direct donations still increase the asset value of all existing shares and can affect rounding, but the reviewed parameters do not expose the usual cheap empty-vault theft pattern.

Impact. Residual effects are economic rounding and intentional value transfer to incumbent LPs, not a confirmed profitable exploit under normal balances.

Recommendation. Retain the virtual offset and add fuzz/property tests for first deposit, donation, total share burn, repeated deposit/withdraw cycles, and extreme balances. Consider adopting an audited ERC-4626-style implementation if broader composability is desired.

I-02 — Arithmetic is checked; extreme values may revert rather than corrupt state — Informational

Location: `assetsOf(), line 108; deposit(), line 127; fillExit(), lines 169 and 171`

Description. Solidity 0.8.24 checks integer overflow and underflow by default. No unchecked blocks are used. Multiplications such as `s * totalAssets()`, `amount * (totalShares + 1000)`, and `gross * 90` can theoretically overflow only at values far above realistic ETH supply. The `used + gross` check likewise reverts safely if parameters are set to pathological values.

Impact. At physically realistic ETH balances there is no practical overflow risk. At synthetic extreme values, affected operations revert, producing denial of service rather than silent accounting corruption.

Recommendation. No immediate change is required. For mathematical robustness and reuse, use full-precision `mulDiv` helpers and apply sane hard caps to owner-set parameters.

5. Best-Practices and Security Checklist

Control	Status	Assessment
SPDX license	**Pass**	MIT identifier is present on line 1.
Version pragma	**Pass**	Solidity ^0.8.24 matches the verified 0.8.24 compiler.
Comments / NatSpec	**Pass**	Purpose and primary flows are documented; expand trust assumptions.
Events	**Mostly pass**	Core actions emit events; ownership transfer lacks an event (L-04).
Access-control modifiers	**Pass with risk**	Owner/relayer checks are correct; key trust is high (H-01/H-02).
Immutables / storage	**Note**	Roles are mutable by design. Parameter bounds are inconsistent (H-02).
Error handling	**Pass**	Custom errors and failed-call checks are used consistently.
Fallback / receive	**Needs improvement**	Payable receive is deliberate but conflates repayments/donations (L-02).
View functions	**Pass with risk**	Views are clear; totalAssets omits in-transit receivables (M-01).
Reentrancy	**Pass / harden**	CEI prevents an identified exploit; add a guard (L-01).
Integer overflow	**Pass**	Solidity checked arithmetic; only unrealistic-value reverts (I-02).
External calls	**Pass / harden**	Return value checked; calls forward gas and may revert (L-01/L-05).
External calls in loops	**Pass**	No external calls occur inside loops; contract has no loops.
Gas / block limit	**Pass**	All operations are O(1); no unbounded iteration.
Randomness	**Not applicable**	The contract uses no randomness.
Denial of service	**Low risk**	Recipient/relayer receive behavior can revert fills (L-05).
Code complexity	**Pass**	Small, dependency-free, and readable; economic model needs stronger documentation.

6. Prioritized Remediation Plan

Before material LP funding. Add verifiable/threshold authorization for exits; enforce immutable cap ceilings; move ownership to a multisig; add a timelock for cap increases and role changes; deploy monitoring and automatic emergency pause procedures.

Before routine operation. Redesign NAV/share accounting for in-transit receivables and replace the calendar-day cap with a rolling limiter. Validate the design with economic invariants and adversarial deposit/fill/repay/withdraw sequences.

Defense in depth. Add reentrancy protection, two-step ownership, explicit repayment reconciliation, receiver-failure handling, and tranche-aware withdrawal locks.

Ongoing assurance. Add fuzz and invariant tests; conduct independent review of the L2 exit contract, bridge, relayer code, monitoring, and key-management controls because those components are outside this audit but determine system safety.

Appendix A — Severity Methodology

Critical. Direct, broadly exploitable loss of most/all funds or permanent protocol failure.

High. Material fund loss or severe control failure, often requiring a privileged-key compromise or specific conditions.

Medium. Meaningful economic loss, accounting distortion, or availability failure under plausible conditions.

Low. Limited impact, hardening gap, operational hazard, or issue requiring unlikely conditions.

Informational. Best practice, design observation, or theoretical edge case without a demonstrated practical exploit.

Appendix B — Limitations and Disclaimer

This audit is a point-in-time, best-effort manual review of the source code explicitly listed in scope. It does not cover all deployed or off-chain components, governance and key custody in practice, economic behavior under every market condition, undisclosed dependencies, compiler defects, or future code and configuration changes. No review can prove the absence of vulnerabilities. Findings and severities involve professional judgment and may change as assumptions or system context become known.

This report is provided for technical risk-reduction purposes only. It is not a warranty, certification, endorsement, or guarantee of security, and it is not financial, legal, tax, or investment advice. Users and operators remain responsible for independent testing, monitoring, incident response, safe key management, and decisions concerning deployment or use. Remediation should be reviewed and retested before production rollout.