Quantstamp

Indigo Vault

Executive Summary

This audit report was prepared by Quantstamp, the leader in blockchain security.

Type	Tokenized Vault
Timeline	2026-03-02 through 2026-03-05
Language	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	README.md
Source Code	• Hiich/indigo-fund #6d3b012
Auditors	• Darren Jensen Auditing Engineer • Roman R Senior Auditing Engineer • Yamen Merhi Auditing Engineer

Documentation quality	Medium
Test quality	High
Total Findings	6 Fixed: 4 Acknowledged: 2
High severity findings	0
Medium severity findings	1 Fixed: 1
Low severity findings	2 Fixed: 2
Undetermined severity findings	0
Informational findings	3 Fixed: 1 Acknowledged: 2

Summary of Findings

The audit of the Indigo Vault Protocol reviewed two contracts: an ERC-4626 compliant vault (`IndigoVault`) and its deployment factory (`IndigoVaultFactory`). The `IndigoVault` is designed to tokenize off-chain yield-bearing positions managed by Psalion. It implements a custom withdrawal queue with a 3-day delay and 7-day expiry (providing a 4-day claim window), admin-controlled NAV updates bounded to 20% per update with a 12-hour cooldown, and a role-based access control system (`DEFAULT_ADMIN_ROLE` , `VAULT_ADMIN` , `GUARDIAN`) governing deposits, liquidity management, and emergency pausing. The `IndigoVaultFactory` provides permissioned deployment and registry functionality for vault instances.

The codebase is well-structured and builds on audited OpenZeppelin v5.1.0 primitives, inheriting protections such as virtual share inflation-attack mitigation, reentrancy guards, and the two-step admin transfer delay. Instant withdrawals and redemptions are explicitly disabled, forcing all exits through the queued withdrawal mechanism. Test coverage is thorough, with explicit validation of edge cases including donation attacks, pause behaviour, and withdrawal queue state transitions.

The NAV updating mechanism allows fixed limits that may hinder responsiveness during volatile market conditions, and it is recommended that parameters like `NAV_UPDATE_COOLDOWN` be made configurable to adapt to market needs. Additionally, the current setup allows for economically front-runnable transactions that could disadvantage users, which demands a reassessment of how withdrawal amounts are calculated ideally shifting to dynamic calculations at the claim time rather than at request time. One low-severity finding identified an operational liveness risk where `updateNAV()` cannot bootstrap `managedAssets` from zero. Two informational findings address edge cases in withdrawal queue mechanics and state growth.

Overall, addressing these vulnerabilities and implementing the suggested recommendations would improve the contract's robustness and user trust significantly.

Fix-Review Update 2026-03-12:
The client has successfully addressed several vulnerabilities in their code with the provided updates. Vulnerabilities IND-1 through IND-4 have been fully fixed, implementing changes that eliminate front-running incentives, ensure NAV freshness checks, allow configurable NAV update cooldowns, and handle edge cases when managed assets are zero. However, vulnerabilities IND-5 and IND-6 have been acknowledged but not fixed, with the client providing rationales for their design choices regarding the withdrawal cancellation window and unbounded growth in certain arrays. Overall, all core vulnerabilities have been rectified, however, some concerns remain acknowledged.

Fix-Review Update 2026-03-23:
In commit `745e56b` , the client made minor adjustments to the `IndigoVault` contract, adding overloaded `deposit()` and `withdraw()` functions that accept an optional `partnerId` parameter, emitting it via new `ReferralDeposit` and `ReferralWithdrawal` events. They also added a `DepositRouter` contract; however, this is marked as `DEPRECATED` and is out of scope for this audit. These changes are minimal and do not affect the final report assessment.

ID	DESCRIPTION	SEVERITY	STATUS
IND-1	Price-per-Share Updates Are Economically Front-Runnable	Medium	Fixed
IND-2	Stale `NAV` May Be Used for Deposits and Withdrawal Requests	Low	Fixed
IND-3	`NAV` Update Bounds Reduce Volatility Responsiveness	Low	Fixed
IND-4	`NAV` Cannot Be Set From Zero, Creating Operational Liveness Risk	Informational	Fixed
IND-5	Cancellation Dead Zone Between Delay and Expiry	Informational	Acknowledged
IND-6	Unbounded Withdrawal History Causes Permanent State Growth	Informational	Acknowledged

Assessment Breakdown

Quantstamp's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

> **Disclaimer**
>
> Only features that are contained within the repositories at the commit hashes specified on the front page of the report are within the scope of the audit and fix review. All features added in future revisions of the code are excluded from consideration in this report.

Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow / underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Methodology

1. Code review that includes the following
    1. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
    2. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
    3. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.
2. Testing and automated analysis that includes the following:
    1. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
    2. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarity, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Scope

Files Included

Repo: `https://github.com/Hiich/indigo-fund`

Included Paths: `packages/contracts/src`

Operational Considerations

- Ownership/Admin Can Be Renounced. Both `IndigoVault` and `IndigoVaultFactory` permit admin/owner renunciation.
- The vault admin (`VAULT_ADMIN`) has full control of setting NAV which directly affects share price and vault economics.
- The vault admin must ensure sufficient on-chain liquidity is available during the withdrawal claim window to satisfy pending withdrawal obligations; otherwise, `claimWithdrawal()` function calls will revert with `InsufficientLiquidity` .
- The vault has a critical off-chain dependency on the Psalion omnibus wallet. Assets swept via `sweep()` leave the contract entirely and are held off-chain. The on-chain `managedAssets` value is self-reported by `VAULT_ADMIN` with no on-chain verification. Users implicitly trust that the admin accurately reflects real off-chain balances.
- `claimWithdrawal()` checks only the contract's live token balance (`balanceOf(address(this))`), not `totalAssets()` . Even if `managedAssets` is non-zero (i.e., funds exist off-chain), claims will revert if on-chain liquidity is insufficient. The admin must proactively call `fundVault()` before the 3-day claim window opens.
- The withdrawal amount (`assetsOwed`) is locked in at `requestWithdrawal()` time based on the NAV at that moment. Subsequent NAV increases do not benefit the withdrawing user — they receive the lower locked-in value. Conversely, NAV decreases after the request are also not reflected.
- Users who fail to claim their withdrawal within the 3–7 day window will have their request expired and shares returned. They must re-initiate a new withdrawal request at the current NAV, which may be less favourable than at the time of the original request.
- `setDepositCap(0)` effectively halts all new deposits without emitting a `Paused` event and without triggering any pause-monitoring infrastructure. This acts as a silent secondary pause mechanism that may be less visible to users and off-chain monitors.
- When `managedAssets` is zero, calling `updateNAV()` with any non-zero value will always revert due to the percentage-change bounds check. The only way to initialise a non-zero `managedAssets` is via `sweep()` . This creates a strict operational ordering requirement.

Key Actors And Their Capabilities

DEFAULT_ADMIN_ROLE

- Can grant/revoke roles
- Can call `setOmnibus()`
- Role transfer is subject to a 2-day timelock (`ADMIN_TRANSFER_DELAY`)

VAULT_ADMIN

- Can increase NAV arbitrarily (inflate share price)
- Can decrease NAV arbitrarily (deflate share price)
- Can `sweep()` — transfers on-chain assets to the omnibus address
- Can `fundVault()` — returns assets from the omnibus address to the vault
- Can `updateNAV()` — bounded to 20% max change per update, with a 12-hour cooldown
- Can `setDepositCap()` - updates the deposit cap

GUARDIAN

- Can `pause()` — blocks new deposits and withdrawal requests
- Can `unpause()` — restores normal vault operation

Depositor

- Can `deposit()` / `mint()` — provide assets in exchange for vault shares
- Can `requestWithdrawal()` — locks shares in the vault and queues an asset claim
- Can `cancelWithdrawal()` — cancels a pending request and recovers shares (only within the 3-day delay window)
- Can `claimWithdrawal()` — redeems assets after the 3-day delay and before the 7-day expiry

Anyone (Permissionless)

- Can call `expireWithdrawal()` on any overdue request (past 7-day expiry) — returns shares to the original requester.

Findings

IND-1 Price-per-Share Updates Are Economically Front-Runnable

Medium Fixed

Update

Marked as "Fixed" by the client.

Addressed in: `c5c10dd` .

The client provided the following explanation:

> Withdrawal assets are now computed at `claimWithdrawal()` time using `convertToAssets(req.shares)` instead of being locked at `requestWithdrawal()` time. The `assetsOwed` field was removed from the `WithdrawalRequest` struct. This eliminates the front-running incentive since users can no longer lock in a favorable exchange rate before NAV updates.

File(s) affected: `packages/contracts/src/IndigoVault.sol`

Description: `IndigoVault` exposes multiple privileged operations that can change the effective price-per-share by mutating `managedAssets`, including `updateNAV()`, `sweep()`, and `fundVault()`, which all impact `totalAssets()` and therefore the ERC4626 exchange rate used by `deposit()/mint()`. In addition, `requestWithdrawal()` uses `convertToAssets()` to lock `assetsOwed` at request time, making user exit terms directly sensitive to the timing of these price-per-share updates.

However, because these functions's transaction appear publicly in the mempool, they can be observed and targeted for ordering advantages. Under the current architecture where `requestWithdrawal()` locks the asset amount at request time, users can front-run or back-run operator updates (especially loss events or large NAV adjustments) by submitting `requestWithdrawal()` or `deposit()` immediately before the update, effectively shifting losses or capturing value at the expense of slower users who do not monitor timing.

This could lead to repeated unfair value transfer around operator-driven pricing updates, where sophisticated users reduce exposure by racing updates while passive users bear more of the loss, resulting in persistent MEV-like extraction and degraded trust in the queue-based exit mechanism.

Recommendation:

- If the design is to keep the current architecture (locking `assetsOwed` at request time), submit price-per-share affecting operations (`updateNAV()`, `sweep()`, and `fundVault()`) using private transaction routing (e.g., Flashbots and MEVBlocker) to reduce mempool observability and prevent users from anticipating update timing.
- Otherwise, recalculate the asset amount at `claimWithdrawal()` time based on the current exchange rate (do not store a fixed `assetsOwed`), which improves fairness by distributing gains/losses according to actual redemption time but introduces exchange-rate risk where users may receive less than they expected at request time.
- Consider having the user specify a slippage of acceptable deviation they accept, otherwise the shares are reverted. This protect users from slippage.
- Alternatively, if the behavior is desired consider documenting it.

IND-2 Stale `NAV` May Be Used for Deposits and Withdrawal Requests

Low · Fixed

Update

Marked as "Fixed" by the client.
Addressed in: `928a5a0`.
The client provided the following explanation:

> Added an opt-in NAV freshness check. A new `maxNAVStaleness` parameter `(default 0 = disabled)` can be set by `VAULT_ADMIN` via `setMaxNAVStaleness()`. When enabled, `deposit()`, `mint()`, `maxDeposit()`, and `maxMint()` revert with `StaleNAV` if `block.timestamp > lastNAVUpdate + maxNAVStaleness`. The check is skipped before the first NAV update (`lastNAVUpdate == 0`). This gives the admin control to enforce freshness without breaking backward compatibility.

File(s) affected: `packages/contracts/src/IndigoVault.sol`

Description: `IndigoVault.deposit()`, `mint()`, and `requestWithdrawal()` rely on the ERC4626 exchange rate derived from `totalAssets()`, which includes `managedAssets` updated via `updateNAV()` and timestamped by `lastNAVUpdate`. The intended flow is that `VAULT_ADMIN` regularly updates NAV so users deposit and lock withdrawal amounts against a reasonably fresh view of the off-chain managed assets.

However, these user entry points do not enforce any freshness requirement on `lastNAVUpdate`, so they can proceed even if NAV has not been updated for an extended period due to operator downtime, misconfiguration, or delayed reporting. This is analogous to using an oracle price without checking its last update time, and it can result in users depositing or locking withdrawal terms against a stale or materially inaccurate exchange rate.

This could lead to unfair outcomes and accounting drift in practice, where users mint shares at stale prices or lock `assetsOwed` based on outdated NAV, especially during volatile periods or when off-chain assets materially change while on-chain reporting lags.

Recommendation:

- Add an optional NAV freshness constraint for `deposit()`, `mint()`, and `requestWithdrawal()` (e.g., `require block.timestamp - lastNAVUpdate <= maxNAVStaleness`, with a configurable threshold).
- Alternatively, if the behavior is desired consider documenting it.

IND-3 `NAV` Update Bounds Reduce Volatility Responsiveness

Low · Fixed

Update

Marked as "Fixed" by the client.
Addressed in: `7484358`.
The client provided the following explanation:

> `NAV_UPDATE_COOLDOWN` is now a configurable state variable `navUpdateCooldown` (default 12 hours, matching original behavior). `VAULT_ADMIN` can adjust it via `setNAVUpdateCooldown()`, bounded between `MIN_NAV_COOLDOWN` (1 hour) and

`MAX_NAV_COOLDOWN` (7 days). This allows faster NAV reporting during volatile periods while maintaining safety rails.

File(s) affected: `packages/contracts/src/IndigoVault.sol`

Description: `IndigoVault.updateNAV()` enforces both a cooldown (`NAV_UPDATE_COOLDOWN`) and a maximum per-update change (`MAX_NAV_CHANGE_BPS`) by comparing the absolute difference between `newManagedAssets` and `oldManagedAssets` against a fixed percentage bound. The intended flow is to smooth share price changes and reduce the risk of abrupt or erroneous NAV updates.

However, during high-volatility market condition there can be legitimate situations where the NAV must be updated by more than 20% or updated more frequently than the fixed cooldown allows. Having a fixed 20% per-transaction bound can be acceptable as a default safety rail, but combining it with an unchangeable cooldown and constant thresholds can prevent timely and accurate NAV reporting when conditions demand faster or larger adjustments.

This could lead to prolonged on-chain mispricing where deposits/mints and queued withdrawals operate against an outdated NAV during volatile periods, creating avoidable fairness and operational issues.

Recommendation: Consider making `NAV_UPDATE_COOLDOWN` configurable so the system can adapt during volatile market conditions when desired.

IND-4

`NAV` Cannot Be Set From Zero, Creating Operational Liveness Risk

• Informational Fixed

Update

Marked as "Fixed" by the client.
Addressed in: `235e58f` .
The client provided the following explanation:

> `updateNAV()` now explicitly handles the `oldManagedAssets == 0` case by skipping the percentage bounds check, allowing bootstrap from zero. The vault no longer requires `sweep()` to be called first. If `managedAssets` returns to zero (e.g., after `fundVault()`), it can be re-initialized via `updateNAV()` without operational workarounds.

File(s) affected: `packages/contracts/src/IndigoVault.sol`

Description: In the `IndigoVault` contract, the `updateNAV()` function enforces a bounded NAV change using the following condition:

```
if (diff * 10000 > oldManagedAssets * MAX_NAV_CHANGE_BPS) {
    revert NAVChangeExceedsLimit();
}
```

When `oldManagedAssets` is zero, the right-hand side of the inequality evaluates to zero. Therefore, any call where `newManagedAssets > 0` results in a `NAVChangeExceedsLimit` revert.

As a result, it is mathematically impossible to set `managedAssets` from zero to a positive value via `updateNAV()` . The only permitted call in this state is `updateNAV(0)` .

The inline comment suggests calling `sweep()` first to seed `managedAssets` . While this operational sequence works in practice, it creates an implicit dependency: the system cannot bootstrap or recover `managedAssets` from zero using `updateNAV()` alone.

Additionally, if `managedAssets` ever returns to zero (for example, after `fundVault()` fully offsets it), the system re-enters this constrained state and `updateNAV()` cannot increase `managedAssets` again unless `sweep()` is executed first.

Although this behavior is not exploitable by external actors, it introduces an operational liveness dependency and could lead to confusion or production incidents if the required sequencing assumption is violated.

Recommendation: Consider making the bootstrap behavior explicit. Possible approaches include:

1. Explicitly handling the `oldManagedAssets == 0` case and allowing a one-time initialization update.
2. Introducing a dedicated initialization function for setting the first positive `managedAssets` value.
3. Keeping the current behavior but adding an explicit guard and clearer error message indicating that `sweep()` must be called before `updateNAV()` when `managedAssets == 0` .

Making this dependency explicit improves clarity and reduces the risk of operational misconfiguration.

IND-5 Cancellation Dead Zone Between Delay and Expiry

• Informational Acknowledged

Update

Marked as "Acknowledged" by the client.
The client provided the following explanation:

> The dead zone (days 3–7) is intentional by design. Allowing cancellation during the claimable window would enable a free-option attack: users could cancel their withdrawal if the NAV moves against them, effectively getting a risk-free bet. NatSpec documentation has been added to `cancelWithdrawal()` explaining this rationale. The frontend and user documentation will also clarify this window.

File(s) affected: `packages/contracts/src/IndigoVault.sol`

Description: `IndigoVault.cancelWithdrawal()` is only allowed before `WITHDRAWAL_DELAY`, while `IndigoVault.expireWithdrawal()` becomes available only after `WITHDRAWAL_EXPIRY`, and both paths ultimately return the user's locked shares. However, in the middle window (`WITHDRAWAL_DELAY` → `WITHDRAWAL_EXPIRY`) cancellation is explicitly disallowed even though the post-expiry outcome is still a shares refund. This can confuse users and integrators unless there is an intentional policy reason for enforcing this dead zone.

Recommendation: Consider documenting the rationale for the cancellation dead zone and surfacing it in user-facing tooling so users and integrators understand that the day 3–7 window is exclusively for claiming, not cancellation

IND-6
Unbounded Withdrawal History Causes Permanent State Growth

Informational · Acknowledged

Update
Marked as "Acknowledged" by the client.
The client provided the following explanation:

> The unbounded growth is accepted by design. Historical withdrawal data is indexed off-chain via the event indexer. The on-chain arrays (`withdrawalQueue` and `_userRequests`) are only used for user-facing lookups, which remain bounded in practice per individual user. No state-changing functions iterate over these arrays, so gas costs are unaffected. NatSpec documentation has been added to `getUserRequests()` explaining this design decision. A pagination helper (`getUserRequestsSlice`) can be added in a future version if needed.

File(s) affected: `packages/contracts/src/IndigoVault.sol`

Description: The contract appends every withdrawal request to `withdrawalQueue` and stores per-user request indices in `_userRequests[user]`. These arrays are never pruned after requests are claimed, cancelled, or expired.

This does not affect core vault accounting or on-chain execution of withdrawals (no iteration over these arrays in state-changing functions). However, it causes permanent state growth and can make `getUserRequests(address)` return very large arrays over time, which may be impractical for frontends/indexers due to RPC response limits and client-side processing overhead.

Recommendation: If long-term scale is a concern, consider:

- Relying on events as the primary source of historical withdrawal data and treating on-chain arrays as minimal state, and/or
- Adding pagination helpers (e.g., `getUserRequestsSlice(user, start, count)`), and/or
- Documenting that historical queue data is expected to be indexed off-chain.

Auditor Suggestions

S1 Missing Input Validation

Acknowledged

Update
Marked as "Acknowledged" by the client.
The client provided the following explanation:

> The `createVault()` function is `onlyOwner`, so only the trusted factory owner can call it. Empty `name_/symbol_` and zero `depositCap_` are valid operational states (e.g., `depositCap_ = 0` effectively pauses deposits at launch, which is a desired capability). Adding validation here would restrict legitimate use cases without meaningful security benefit since the caller is already permissioned.

File(s) affected: `packages/contracts/src/IndigoVaultFactory.sol`

Description: The following function parameters are used without prior bounds checking:

1. `IndigoVaultFactory.createVault()`:
   1. `name_` and `symbol_` are not checked to be non-zero length strings.
   2. `depositCap_`: Not checked to be different from zero or otherwise unbound.
2. `IndigoVault.constructor()`: `depositCap_` not checked to be different from zero or otherwise unbound.

S2 Ownership Can Be Renounced

Fixed

> **Update**
> Marked as "Fixed" by the client.
> Addressed in: `7cdd474`.
> The client provided the following explanation:
>
> > `renounceOwnership()` has been overridden and disabled in `IndigoVaultFactory` to prevent accidental lockout. For `IndigoVault`, the contract uses `AccessControlDefaultAdminRules` (OZ) which already enforces a 2-day timelock on admin transfers — the admin role cannot be accidentally renounced in a single transaction, which provides sufficient protection.

File(s) affected: `packages/contracts/src/IndigoVaultFactory.sol`, `packages/contracts/src/IndigoVault.sol`

Description: If the owner renounces their ownership, all ownable contracts will be left without an owner. Consequently, any function guarded by the `onlyOwner` modifier will no longer be able to be executed.

Recommendation: Confirm that this is the intended behavior. If not, override and disable the `renounceOwnership()` function in the affected contracts. For extra security, consider using a two-step process when transferring the ownership of the contract (e.g. `Ownable2Step` from OpenZeppelin).

S3 Auditor Suggestions

Acknowledged

> **Update**
> Marked as "Fixed" by the client.
> Addressed in: `7cdd474`.

File(s) affected: `packages/contracts/src/IndigoVault.sol`, `packages/contracts/src/IndigoVaultFactory.sol`

Description: The following auditor suggestions and best practices have been identified:

1. `Fixed` In the `IndigoVault.sol#L400` the comment mentions `>10%` change. However, `MAX_NAV_CHANGE_BPS` is `2000` and thereby `20%`.
2. `Unresolved` In the `IndigoVault` contract the `OmnibusUpdated` event has `address` parameters `oldOmnibus` and `newOmnibus` which are not indexed.
3. `Unresolved` In the `IndigoVault` contract, consider adding a check inside `deposit()` and `mint()` functions to revert if the resulting shares/assets would be zero.
4. `Unresolved` In the `IndigoVaultFactory` contract the `createVault()` function performs redundant zero-address checks for `asset_`, `guardian`, and `omnibus` that are also validated in the `IndigoVault` constructor. Consider removing the redundant checks from the factory to reduce duplication.
5. `Unresolved` Given most of the planned vaults use low-decimal assets (6 or 8 decimals), consider overriding `_decimalsOffset()` to 3 for stronger inflation attack protection.

Definitions

- **High severity** – High-severity issues usually put a large number of users' sensitive information at risk, or are reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
- **Medium severity** – Medium-severity issues tend to put a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or are reasonably likely to lead to moderate financial impact.
- **Low severity** – The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
- **Informational** – The issue does not pose an immediate risk, but is relevant to security best practices or Defence in Depth.
- **Undetermined** – The impact of the issue is uncertain.
- **Fixed** – Adjusted program implementation, requirements or constraints to eliminate the risk.

- **Mitigated** – Implemented actions to minimize the impact or likelihood of the risk.
- **Acknowledged** – The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).

Appendix

File Signatures

The following are the SHA-256 hashes of the reviewed files. A file with a different SHA-256 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different SHA-256 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review.

Files

Repo: `https://github.com/Hiich/indigo-fund`

- `106...231 ./packages/contracts/src/IndigoVault.sol`
- `b6a...924 ./packages/contracts/src/IndigoVaultFactory.sol`

Test Suite Results

```
forge test
```

```
Ran 19 tests for test/IndigoVaultFactory.t.sol:IndigoVaultFactoryTest
[PASS] test_createMultipleVaults_sameAsset() (gas: 5743613)
[PASS] test_createVault_emitsEvent() (gas: 2896266)
[PASS] test_createVault_nonOwner_reverts() (gas: 23695)
[PASS] test_createVault_success() (gas: 2908378)
[PASS] test_createVault_zeroAdmin_reverts() (gas: 20961)
[PASS] test_createVault_zeroAsset_reverts() (gas: 20882)
[PASS] test_createVault_zeroGuardian_reverts() (gas: 21036)
[PASS] test_createVault_zeroOmnibus_reverts() (gas: 21067)
[PASS] test_getVaults_empty() (gas: 8462)
[PASS] test_getVaults_returnsCorrectList() (gas: 5747755)
[PASS] test_isVault_returnsFalse_forRandomAddress() (gas: 7903)
[PASS] test_isVault_returnsFalse_forZeroAddress() (gas: 7848)
[PASS] test_isVault_returnsTrue_forCreatedVaults() (gas: 2893177)
[PASS] test_owner_isSetCorrectly() (gas: 10008)
[PASS] test_transferOwnership_nonPendingOwner_reverts() (gas: 42809)
[PASS] test_transferOwnership_twoStep() (gas: 33260)
[PASS] test_vaultCount_incrementsOnCreate() (gas: 5746707)
[PASS] test_vaultCount_initiallyZero() (gas: 7691)
[PASS] test_vaults_publicArrayAccessor() (gas: 2893463)
Suite result: ok. 19 passed; 0 failed; 0 skipped; finished in 207.81ms (469.43ms CPU time)

Ran 107 tests for test/IndigoVault.t.sol:IndigoVaultTest
[PASS] testFuzz_deposit_withdraw_roundTrip(uint256) (runs: 256, μ: 307186, ~: 307270)
[PASS] testFuzz_navUpdate_withinBounds(uint256) (runs: 256, μ: 198236, ~: 198264)
[PASS] test_adminTransferDelay_constant() (gas: 5772)
[PASS] test_adminTransfer_canCancel() (gas: 28738)
[PASS] test_adminTransfer_twoStep() (gas: 59495)
[PASS] test_cancelWithdrawal_afterDelay_reverts() (gas: 328063)
[PASS] test_cancelWithdrawal_alreadyCancelled_reverts() (gas: 295955)
[PASS] test_cancelWithdrawal_alreadyClaimed_reverts() (gas: 301866)
[PASS] test_cancelWithdrawal_invalidIndex_reverts() (gas: 18225)
[PASS] test_cancelWithdrawal_notOwner_reverts() (gas: 329119)
[PASS] test_cancelWithdrawal_success() (gas: 299520)
[PASS] test_claimWithdrawal_after72Hours() (gas: 308571)
[PASS] test_claimWithdrawal_afterExpiry_reverts() (gas: 328370)
[PASS] test_claimWithdrawal_alreadyClaimed_reverts() (gas: 301863)
[PASS] test_claimWithdrawal_before72Hours_reverts() (gas: 328206)
[PASS] test_claimWithdrawal_exactlyAt72Hours() (gas: 298862)
[PASS] test_claimWithdrawal_insufficientLiquidity_reverts() (gas: 386177)
```

```
[PASS] test_claimWithdrawal_invalidIndex_reverts() (gas: 18268)
[PASS] test_claimWithdrawal_justBeforeExpiry_succeeds() (gas: 298818)
[PASS] test_claimWithdrawal_notOwner_reverts() (gas: 329623)
[PASS] test_constants() (gas: 8650)
[PASS] test_constructor_zeroAdmin_reverts() (gas: 99094)
[PASS] test_constructor_zeroAsset_reverts() (gas: 187060)
[PASS] test_constructor_zeroGuardian_reverts() (gas: 192157)
[PASS] test_constructor_zeroOmnibus_reverts() (gas: 192234)
[PASS] test_deposit_exactCap_succeeds() (gas: 118383)
[PASS] test_deposit_exceedsCap_reverts() (gas: 43736)
[PASS] test_deposit_multipleUsers() (gas: 157124)
[PASS] test_deposit_success() (gas: 115539)
[PASS] test_deposit_whenPaused_reverts() (gas: 40793)
[PASS] test_deposit_zeroAmount_reverts() (gas: 18531)
[PASS] test_donationAttack_firstDepositor() (gas: 242725)
[PASS] test_expireWithdrawal_alreadyCancelled_reverts() (gas: 298410)
[PASS] test_expireWithdrawal_alreadyClaimed_reverts() (gas: 304180)
[PASS] test_expireWithdrawal_notExpired_reverts() (gas: 329923)
[PASS] test_expireWithdrawal_success() (gas: 303862)
[PASS] test_fullFlow_depositSweepNAVWithdrawFundClaim() (gas: 485931)
[PASS] test_fundVault_exceedsManaged_reverts() (gas: 231311)
[PASS] test_fundVault_nonAdmin_reverts() (gas: 13497)
[PASS] test_fundVault_pullsTokensFromCaller() (gas: 199809)
[PASS] test_fundVault_success() (gas: 197070)
[PASS] test_fundVault_withoutApproval_reverts() (gas: 208735)
[PASS] test_fundVault_withoutTokens_reverts() (gas: 205156)
[PASS] test_fundVault_zeroAmount_reverts() (gas: 18558)
[PASS] test_getUserRequests() (gas: 448145)
[PASS] test_getUserRequests_emptyForNewUser() (gas: 10665)
[PASS] test_getWithdrawalRequest_invalidIndex_reverts() (gas: 11034)
[PASS] test_grantRole_defaultAdmin_reverts() (gas: 13729)
[PASS] test_maxDeposit_returnsFullCap_initially() (gas: 20020)
[PASS] test_maxDeposit_returnsRemainingCap() (gas: 111620)
[PASS] test_maxDeposit_returnsZero_whenCapReached() (gas: 118997)
[PASS] test_maxDeposit_returnsZero_whenPaused() (gas: 39562)
[PASS] test_maxMint_returnsRemainingShares() (gas: 113714)
[PASS] test_maxMint_returnsZero_whenCapReached() (gas: 118952)
[PASS] test_maxMint_returnsZero_whenPaused() (gas: 39604)
[PASS] test_maxRedeem_returnsZero() (gas: 109943)
[PASS] test_maxWithdraw_returnsZero() (gas: 109930)
[PASS] test_mint_exceedsCap_reverts() (gas: 47926)
[PASS] test_mint_success() (gas: 114602)
[PASS] test_mint_whenPaused_reverts() (gas: 40791)
[PASS] test_mint_zeroShares_reverts() (gas: 18507)
[PASS] test_multipleWithdrawalRequests_differentUsers() (gas: 530785)
[PASS] test_navDecrease_sharesLoseValue() (gas: 275820)
[PASS] test_navIncrease_affectsNewDepositors() (gas: 288959)
[PASS] test_pause_byGuardian() (gas: 37427)
[PASS] test_pause_nonGuardian_reverts() (gas: 13370)
[PASS] test_pendingCount() (gas: 450980)
[PASS] test_pendingCounters_trackCorrectly() (gas: 457641)
[PASS] test_previewRedeem_reverts() (gas: 8535)
[PASS] test_previewWithdraw_reverts() (gas: 8625)
[PASS] test_redeem_reverts() (gas: 110972)
[PASS] test_requestWithdrawal_insufficientShares_reverts() (gas: 114037)
[PASS] test_requestWithdrawal_success() (gas: 330755)
[PASS] test_requestWithdrawal_whenPaused_reverts() (gas: 138102)
[PASS] test_requestWithdrawal_zeroShares_reverts() (gas: 113819)
[PASS] test_roles_assignedCorrectly() (gas: 21728)
[PASS] test_setDepositCap_nonAdmin_reverts() (gas: 13399)
[PASS] test_setDepositCap_success() (gas: 22278)
[PASS] test_setOmnibus_nonAdmin_reverts() (gas: 15148)
[PASS] test_setOmnibus_selfAddress_reverts() (gas: 13567)
[PASS] test_setOmnibus_success() (gas: 185930)
[PASS] test_setOmnibus_zeroAddress_reverts() (gas: 13476)
[PASS] test_shareTransfer_thenWithdraw() (gas: 326101)
[PASS] test_sixDecimalToken_deposit_withdraw() (gas: 3581047)
[PASS] test_sweep_exactlyAtPendingLimit_succeeds() (gas: 385133)
[PASS] test_sweep_goesToOmnibus() (gas: 174155)
[PASS] test_sweep_liquidityGuard_reverts() (gas: 332220)
[PASS] test_sweep_nonAdmin_reverts() (gas: 112982)
[PASS] test_sweep_success() (gas: 179344)
```

```
[PASS] test_sweep_zeroAmount_reverts() (gas: 18559)
[PASS] test_totalPendingAssetsAmount() (gas: 475421)
[PASS] test_totalPendingAssetsAmount_empty() (gas: 7695)
[PASS] test_unpause_byGuardian() (gas: 28235)
[PASS] test_unpause_nonGuardian_reverts() (gas: 42452)
[PASS] test_updateNAV_changesSharePrice() (gas: 214941)
[PASS] test_updateNAV_cooldown_afterWait_succeeds() (gas: 204709)
[PASS] test_updateNAV_cooldown_tooFrequent_reverts() (gas: 198845)
[PASS] test_updateNAV_emitsEvent() (gas: 201042)
[PASS] test_updateNAV_exactlyAtLimit_succeeds() (gas: 205788)
[PASS] test_updateNAV_exceedsLimit_decrease_reverts() (gas: 172315)
[PASS] test_updateNAV_exceedsLimit_increase_reverts() (gas: 172317)
[PASS] test_updateNAV_firstUpdate_noCooldown() (gas: 197291)
[PASS] test_updateNAV_fromZero_reverts() (gas: 17973)
[PASS] test_updateNAV_nonAdmin_reverts() (gas: 13400)
[PASS] test_updateNAV_success() (gas: 199310)
[PASS] test_withdraw_reverts() (gas: 110916)
[PASS] test_withdrawalQueueLength() (gas: 448116)
Suite result: ok. 107 passed; 0 failed; 0 skipped; finished in 275.95ms (737.70ms CPU time)

Ran 2 test suites in 308.94ms (483.77ms CPU time): 126 tests passed, 0 failed, 0 skipped (126 total tests)
```

Code Coverage

The test coverage was executed using:

```
forge coverage --no-match-coverage "mock|libraries"
```

The coverage is very high and covers almost all branches.

File	% Lines	% Statements	% Branches	% Funcs
src/IndigoVault.sol	97.83% (135/138)	97.66% (167/171)	94.87% (37/39)	100.00% (26/26)
src/IndigoVaultFactory.sol	100.00% (14/14)	100.00% (16/16)	100.00% (4/4)	100.00% (3/3)
Total	98.03% (149/152)	97.86% (183/187)	95.35% (41/43)	100.00% (29/29)

Changelog

- 2026-03-09 - Initial report

About Quantstamp

Quantstamp is a global leader in blockchain security. Founded in 2017, Quantstamp’s mission is to securely onboard the next billion users to Web3 through its best-in-class Web3 security products and services.

Quantstamp’s team consists of cybersecurity experts hailing from globally recognized organizations including Microsoft, AWS, BMW, Meta, and the Ethereum Foundation. Quantstamp engineers hold PhDs or advanced computer science degrees, with decades of combined experience in formal verification, static analysis, blockchain audits, penetration testing, and original leading-edge research.

To date, Quantstamp has performed more than 1300 audits and secured over $200 billion in digital asset risk from hackers. Quantstamp has worked with a diverse range of customers, including startups, category leaders and financial institutions. Brands that Quantstamp has worked with include Ethereum 2.0, Binance, Visa, PayPal, Solana, Canton, Polymarket, PandcakeSwap, Virtuals Protocol, Wayfinder, Lido, TrustWallet, Alchemy, World Economic Forum.

Quantstamp’s collaborations and partnerships showcase our commitment to world-class research, development and security. We're honored to work with some of the top names in the industry and proud to secure the future of web3.

Notable Collaborations & Customers:

- Blockchains: Ethereum 2.0, Solana, Polygon, TON, Cardano, Binance Smart Chain, Avalanche, Arbitrum, Flow
- DeFi: Lido, Ethena, Compound, Curve, Venus, PancakeSwap, Polymarket
- Infra/Staking: TrustWallet, Alchemy, Liquid Collective, Kiln, Galxe, API3, ssv.network, Luganodes
- Immersive: Virtuals Protocol, Wayfinder, OpenSea, Square Enix, Parallel, Camp, Decentraland
- Institutional: Visa, Circle, Revolut, Anthropic, OpenAI, Canton, Republic, Arkham, Sequoia