



Security Review For Anchored



Collaborative Audit Prepared For:
Lead Security Expert(s):

Anchored
0xeix

Date Audited:

vinica_boy
March 10 - March 23, 2026

Introduction

Anchored is the global digital operating system for RWAs. Anchored's first product enables the issuance, redemption and distribution of tokenized public equities on blockchain rails. The protocol's design is focused on facilitating institutional-grade liquidity, while underlying assets are purchased and custodied via a FINRA-regulated broker-dealer in order to ensure 1:1 backing.

The audited contracts govern the complete on-chain minting, burning, and settlement layer for Anchored's tokenized equities stack. This covers router-based user entry, cash and stock balance accounting, order placement and cancellation, mint/burn-based settlement of tokenized shares, cross-chain fund movement between the cashier chain and broker-side gateway, and compliance enforcement for permissioned flows.

Scope

Repository: [AnchoredLabs/anchored-contracts](https://github.com/AnchoredLabs/anchored-contracts)

Audited Commit: [318bc4b4f4222b692aff3f4fb7de32301e8a58d2](https://github.com/AnchoredLabs/anchored-contracts/commit/318bc4b4f4222b692aff3f4fb7de32301e8a58d2)

Final Commit: [375e2d0781d03419142a3d178893a8537218d201](https://github.com/AnchoredLabs/anchored-contracts/commit/375e2d0781d03419142a3d178893a8537218d201)

Files:

- [contracts/AncBrokerGateway.sol](#)
- [contracts/AncCashier.sol](#)
- [contracts/AncOrderSettler.sol](#)
- [contracts/AncStockExchange.sol](#)
- [contracts/AncStockRouter.sol](#)
- [contracts/base/CashierBase.sol](#)
- [contracts/base/CashSettlerBase.sol](#)
- [contracts/base/OrderSettlerBase.sol](#)
- [contracts/base/StockExchangeBase.sol](#)
- [contracts/base/StockRouterBase.sol](#)
- [contracts/bridge/AncBridgeAdapterAcross.sol](#)
- [contracts/bridge/AncBridgeAdapterCctp.sol](#)
- [contracts/compliance/AncCompliance.sol](#)
- [contracts/monday/MondayCashier.sol](#)
- [contracts/monday/MondayOrderSettler.sol](#)
- [contracts/monday/MondayStockExchange.sol](#)

- contracts/monday/MondayStockRouter.sol
- contracts/settler/CashSettler.sol
- contracts/stock/AncStockTokenFactory.sol
- contracts/stock/AncStockToken.sol
- contracts/utils/Create3Deployer.sol
- contracts/utils/LibFundTransfer.sol
- contracts/utils/ProxyAdmin2Step.sol

Final Commit Hash

375e2d0781d03419142a3d178893a8537218d201

Findings

Each issue has an assigned severity:

- High issues are directly exploitable security vulnerabilities that need to be fixed.
- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- Low/Info issues are non-exploitable, informational findings that do not pose a security risk or impact the system's integrity. These issues are typically cosmetic or related to compliance requirements, and are not considered a priority for remediation.

Issues Found

High	Medium	Low/Info
0	1	9

Issues Not Fixed and Not Acknowledged

High	Medium	Low/Info
0	0	0

Issue M-1: The `tif` field is not respected when executing the orders [RESOLVED]

Source: <https://github.com/sherlock-audit/2026-03-anchored-mar-9th-2026/issues/24>

Summary

`StockRouterBase` / `StockExchangeBase` expose a full `TimeInForce` model (DAY, GTC, OPG, IOC, FOK, GTX, GTD, CLS), but the exchange never enforces `tif` during settlement or cancellation. As a result, limit orders are effectively treated as generic persistent orders regardless of the selected TIF type.

Vulnerability Detail

The router forwards `tif` into `placeLimitOrder()`, and the exchange stores it inside `LimitOrder`:

<https://github.com/sherlock-audit/2026-03-anchored-mar-9th-2026/blob/main/anchored-contracts/contracts/base/StockExchangeBase.sol#L283-291>

```
_limitOrders[orderId] = LimitOrder({
    tokenIndex: tokenId,
    quantity: quantity,
    price: price,
    tif: tif,
    cancelRequested: false,
    mintFeeRateSnapshot: _mintFeeRate,
    protocolFeeRateSnapshot: _protocolFeeRate
});
```

However, no later code path reads `o.tif` or applies any TIF-specific behavior. So the current logic effectively treats every limit order as "live until settled or manually cancel-requested". This means that the listed `TimeInForce` modes may be enforceable by some offchain component/matching engine, but they are not enforced onchain by `StockExchangeBase`. However, it's currently not possible to have several modes supported even with offchain components present such as:

- GTD (effectively not supported as no expiry timestamp is currently possible to provide and store inside of the `LimitOrder` struct)
- DAY (no trading-day/session boundary is stored)
- GTC (closest to current behavior, but even here cancellation is not binding, since a cancel request only sets `cancelRequested` and normal settlement can still proceed)

Other types can be theoretically supported if there's an existing offchain component but, if not, the exchange contract has to have onchain guarantees: OPG/CLS (no market

open / close state or auction window exists onchain but offchain logic can only include them in opening/closing auction logic), GTX (no post-only / crossing check exists but offchain engine can reject if it would cross immediately), FOK (no validation onchain but the settler can only submit settlement if full size is available), IOC (can match immediately, cancel remainder offchain).

Impact

Even if the intended design is that the broker / settler is trusted and all matching rules are offchain, some modes such as GTD (and some others previously listed that need to have strong timing boundaries/guarantees) can't be currently supported at all as each order needs its own expiry time/date that's not present in the `LimitOrder` field and can't be fetched during order execution. Only the `deadline` is checked at the submission time:

<https://github.com/sherlock-audit/2026-03-anchored-mar-9th-2026/blob/main/anchored-contracts/contracts/base/StockRouterBase.sol#L97>

```
_requireNotExpired(deadline);
```

but it's generally applied to all the orders as a sanity check.

Tool Used

Manual Review

Recommendation

Consider implementing additional logic for GTD, DAY, GTC types onchain and document the behavior regarding all other modes to clarify how they're supposed to be handled (if the values are offchain policies and not onchain guarantees).

Issue L-1: Price precision mismatch between on-chain limit orders and the underlying broker causes permanent settlement failure [RESOLVED]

Source: <https://github.com/sherlock-audit/2026-03-anchored-mar-9th-2026/issues/25>

Summary

Limit order prices are stored on-chain in WAD (18 decimals) without truncation, but the underlying broker only accepts up to 2 decimal places (4 for prices below \$1). This precision mismatch causes settlement reverts for orders placed with sub-cent precision.

Vulnerability Detail

When a user places a limit order via `placeLimitOrder`, the `price` parameter is stored as-is with full 18-decimal precision. However, the off-chain operator must truncate this price before submitting to the underlying broker, which only supports 2 decimal places.

During settlement, the fill price (truncated) is compared against the stored price (untruncated):

- For sell orders: `fillPrice < o.price` reverts – a sell at \$150.123 truncated to \$150.12 will always fail since $150.12e18 < 150.123e18$.
- For buy orders: `fillPrice > o.price` reverts – a buy at \$150.123 truncated to \$150.13 will always fail since $150.13e18 > 150.123e18$.

Orders placed with prices that have more precision than the underlying broker supports become unseizable.

Impact

Limit orders with sub-cent price precision are permanently unseizable and must be cancelled by the backend. This creates a griefing vector where users can place such orders at no cost – the operator spends gas to cancel them while no fees are collected since the fill amount is zero.

Code Snippet

<https://github.com/sherlock-audit/2026-03-anchored-mar-9th-2026/blob/main/anchored-contracts/contracts/base/StockExchangeBase.sol#L253-L294>

Tool Used

Manual Review

Recommendation

Truncate the limit price on-chain at order placement time to ensure the stored price is always compatible with the underlying broker's precision.

Issue L-2: Permanently failed deposits leave stale pending state in Cashier with no cleanup mechanism [RESOLVED]

Source: <https://github.com/sherlock-audit/2026-03-anchored-mar-9th-2026/issues/26>

Summary

When a deposit permanently fails on the underlying broker side and funds are recovered via `recoverFailedDeposit()`, the corresponding Cashier state (`pending`, `totalPending`, `tokenAccounting.collected`) is never cleared, leaving stale accounting data.

Vulnerability Detail

The deposit flow updates Cashier state in `_queueDeposit` incrementing `pending[user][token].deposit` and `totalPending[token].deposit`. These values are only decremented in `settleDeposits`, which also requires `tokenAccounting[token].collected` to cover the amount.

When a deposit permanently fails on the underlying broker side (e.g. compliance rejection), `recoverFailedDeposit()` on the Gateway sends the funds to a recipient but does not call back into the Cashier to reverse the queued deposit state. There is no mechanism to clear:

- `pending[user][token].deposit`
- `totalPending[token].deposit`
- `tokenAccounting[token].collected`

On-chain operations are not directly blocked – the protocol can continue functioning. However, these variables feed into off-chain backend/UI logic and affect accounting views.

Impact

Permanently failed deposits leave stale state in the Cashier contract. The `pending` and `totalPending` values remain inflated with no way to clear them, polluting accounting data used by off-chain systems.

Code Snippet

<https://github.com/sherlock-audit/2026-03-anchored-mar-9th-2026/blob/main/anchored-contracts/contracts/base/CashierBase.sol#L224-L229>

<https://github.com/sherlock-audit/2026-03-anchored-mar-9th-2026/blob/main/anchored-contracts/contracts/base/CashierBase.sol#L224-L229>

[red-contracts/contracts/base/CashierBase.sol#L343-L384](#)

<https://github.com/sherlock-audit/2026-03-anchored-mar-9th-2026/blob/main/anchored-mar-9th-2026-contracts/contracts/base/CashierBase.sol#L343-L384>

[red-contracts/contracts/AncBrokerGateway.sol#L273-L310](#)

Tool Used

Manual Review

Recommendation

Implement a function to clear Cashier deposit state for permanently failed deposits - decrementing `pending[user][token].deposit`, `totalPending[token].deposit`, and adjusting `tokenAccounting[token].collected`.

Issue L-3: Order IDs lack chain identifier, enabling cross-chain collisions in backend processing [RESOLVED]

Source: <https://github.com/sherlock-audit/2026-03-anchored-mar-9th-2026/issues/27>

Summary

Order IDs generated by `_generateOpId` do not include a chain identifier, so independent Exchange deployments on Monad and Ethereum can produce identical order IDs for different orders.

Vulnerability Detail

`_generateOpId` packs `{operationCounter, tokenId, user}` into a `bytes32`. Since each chain has its own independent Exchange deployment with its own `operationCounter` starting from zero, two orders from different chains can share the same ID e.g. the first order by the same user for the same token on both Monad and Ethereum will produce an identical `bytes32`.

On-chain this is not an issue – each chain's contracts operate independently and never reference each other's order IDs. However, if the backend uses these order IDs to interact with the underlying broker's `client_order_id`, without a chain discriminator, the backend must implement its own disambiguation to avoid confusing orders across chains.

Impact

The backend could misroute or misprocess order operations (settlement, cancellation) if it relies solely on the on-chain order ID without additional chain context when communicating with the underlying broker.

Code Snippet

<https://github.com/sherlock-audit/2026-03-anchored-mar-9th-2026/blob/main/anchored-contracts/contracts/base/StockExchangeBase.sol#L623-L630>

Tool Used

Manual Review

Recommendation

Include `block.chainid` in the order ID packing to make IDs globally unique across chains, or ensure the backend prefixes/namespaces order IDs by chain when submitting to the

Issue L-4: Cancelled limit orders with zero fill amount charge no fees, enabling griefing [RESOLVED]

Source: <https://github.com/sherlock-audit/2026-03-anchored-mar-9th-2026/issues/28>

Summary

Limit orders that are cancelled with zero fill amount incur no fees, allowing users to spam the system with orders that will never execute (unrealistic prices, unfavorable TIF) at no cost while the backend must spend gas to cancel them.

Vulnerability Detail

When a limit order is cancelled via `settleLimitBuyCancels` or `settleLimitSellCancels` with `filledQuantity = 0`, the fee calculation returns zero:

- For buy cancels: `_calculateBuySettlementFees` computes fees based on `executedNotionalUsd` which is 0, so both `_calculateMintFee` and `_calculateProtocolFee` return 0 (early return when `notionalUsd == 0`). The user receives a full refund of `totalLocked`.
- For sell cancels: `_calculateProtocolFee` similarly returns 0 when `usdProceeds = 0`, and all escrowed stock is returned.

A user can place limit orders with unrealistic prices (e.g. buy at \$0.01 for a \$150 stock) or use TIF types like IOC/FOK that are likely to expire unfilled. These orders pass all on-chain validation – they meet `_minOrderValue` requirements since `price * quantity` can be made arbitrarily large – but will never execute on the underlying broker. The backend is forced to process and eventually cancel each order, spending gas, while collecting no fees.

Impact

The backend bears gas costs for cancelling spam orders while the protocol collects zero fees. This creates an asymmetric griefing vector where the cost to the attacker (only the deposit that gets fully refunded) is much lower than the cost to the protocol (gas for cancel transactions).

Code Snippet

<https://github.com/sherlock-audit/2026-03-anchored-mar-9th-2026/blob/main/anchored-contracts/contracts/base/StockExchangeBase.sol#L661-L666>

<https://github.com/sherlock-audit/2026-03-anchored-mar-9th-2026/blob/main/anchored-contracts/contracts/base/StockExchangeBase.sol#L668-L671>

<https://github.com/sherlock-audit/2026-03-anchored-mar-9th-2026/blob/main/anchored-contracts/contracts/base/StockExchangeBase.sol#L431-L442>

<https://github.com/sherlock-audit/2026-03-anchored-mar-9th-2026/blob/main/anchored-contracts/contracts/base/StockExchangeBase.sol#L501-L506>

Tool Used

Manual Review

Recommendation

Introduce a fixed minimum cancellation fee deducted from the user's refund (buy orders) or escrowed stock value (sell orders) regardless of fill amount, to make spam orders economically unviable.

Issue L-5: uint32 minAmount cannot enforce meaningful minimum for high-decimal tokens [RESOLVED]

Source: <https://github.com/sherlock-audit/2026-03-anchored-mar-9th-2026/issues/29>

Summary

`TokenConfig.minAmount` is typed as `uint32`, which cannot represent a meaningful minimum deposit/withdrawal amount for tokens with 18 decimals.

Vulnerability Detail

The `minAmount` field in `TokenConfig` is `uint32`, capping its maximum value at $\sim 4.29e9$. Since `minAmount` is denominated in token decimals, this works for 6-decimal tokens like USDC (max $\sim \$4,294$), but for 18-decimal tokens the maximum representable value is $\sim 4.29e-9$ tokens which is effectively zero. The `minAmount` check in `deposit` and `withdraw` becomes meaningless, unable to enforce any practical minimum.

Impact

If an 18-decimal token is added to the Cashier, the `minAmount` guard cannot enforce a meaningful minimum deposit or withdrawal threshold, bypassing the intended spam/dust protection.

Code Snippet

<https://github.com/sherlock-audit/2026-03-anchored-mar-9th-2026/blob/main/anchored-contracts/contracts/interfaces/ICashier.sol#L9> <https://github.com/sherlock-audit/2026-03-anchored-mar-9th-2026/blob/main/anchored-contracts/contracts/base/CashierBase.sol#L131> <https://github.com/sherlock-audit/2026-03-anchored-mar-9th-2026/blob/main/anchored-contracts/contracts/base/CashierBase.sol#L167>

Tool Used

Manual Review

Recommendation

Increase `minAmount` to a wider type such as `uint96` to support tokens with higher decimal precision.

Issue L-6: Instant settlement threshold bypassable by splitting into multiple transactions [ACKNOWLEDGED]

Source: <https://github.com/sherlock-audit/2026-03-anchored-mar-9th-2026/issues/30>

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

The instant deposit/withdrawal threshold ($\text{buffer} / \text{instantThresholdDivisor}$) is a per-transaction limit that can be bypassed by splitting a large operation into multiple smaller ones, allowing a user to drain a significant portion of the buffer across sequential transactions.

Vulnerability Detail

Instant deposits require $\text{credit} \leq \text{creditBuffer} / \text{instantThresholdDivisor}$ (default divisor = 10, i.e. max 10% per tx). The same logic applies to instant withdrawals via $\text{withdrawalBuffer} / \text{instantThresholdDivisor}$. Since the threshold is recalculated from the current buffer on every call, a user can repeatedly deposit/withdraw just under the threshold:

- $\text{creditBuffer} = 1000$, threshold = 100 → deposit 100, buffer becomes 900
- threshold = 90 → deposit 90, buffer becomes 810
- threshold = 81 → deposit 81, buffer becomes 729
- ...and so on

After ~20 transactions the buffer is reduced to ~12% of its original value. The same pattern applies to the withdrawal buffer. This allows a single user to consume the majority of the buffer capacity meant for all users, potentially forcing subsequent users into the slower queued settlement path.

Impact

A user can drain most of the instant deposit or withdrawal buffer through sequential transactions, degrading instant settlement availability for other users.

Code Snippet

<https://github.com/sherlock-audit/2026-03-anchored-mar-9th-2026/blob/main/anchored-contracts/contracts/base/CashierBase.sol#L143-L149>

<https://github.com/sherlock-audit/2026-03-anchored-mar-9th-2026/blob/main/anchored-contracts/contracts/base/CashierBase.sol#L178-L183>

Tool Used

Manual Review

Recommendation

Introduce a per-user or time-based rate limit on instant operations in addition to the per-transaction threshold, to prevent a single user from consuming a disproportionate share of the buffer.

Issue L-7: Non-unique bridge adapter transferId when identical parameters in same block [RESOLVED]

Source: <https://github.com/sherlock-audit/2026-03-anchored-mar-9th-2026/issues/31>

Summary

Bridge adapter transfer IDs are computed from `keccak256(block.timestamp, token, amount, destChainId, recipient)` without a nonce, so two bridge operations with identical parameters in the same block produce the same ID.

Vulnerability Detail

Both `AncBridgeAdapterAcross` and `AncBridgeAdapterCctp` generate transfer IDs as:

```
transferId = keccak256(abi.encodePacked(block.timestamp, token, amount,
    ↪ destChainId, recipient));
```

This hash does not include a counter or nonce. If two bridge operations occur in the same block with the same token, amount, destination chain, and recipient, they produce an identical `transferId`. This is plausible when the Gateway batches multiple fund transfers of the same size to the same destination.

On-chain this is not an issue – the Gateway tracks its own transfer IDs via `LibFundTransfer.pack()` which includes a unique counter, and the adapter's returned `transferId` is not stored or used by the Gateway. It is only emitted in the `BridgeInitiated` event.

However, if the backend relies on the adapter-level `transferId` from `BridgeInitiated` events to correlate or track bridge operations, collisions would make it impossible to distinguish between two separate transfers.

Impact

Off-chain systems that index `BridgeInitiated` events by `transferId` cannot distinguish duplicate bridge operations within the same block, potentially causing misattribution of bridge settlement status.

Code Snippet

<https://github.com/sherlock-audit/2026-03-anchored-mar-9th-2026/blob/main/anchored-contracts/contracts/bridge/AncBridgeAdapterAcross.sol#L120>

<https://github.com/sherlock-audit/2026-03-anchored-mar-9th-2026/blob/main/anchored-contracts/contracts/bridge/AncBridgeAdapterCctp.sol#L96>

Tool Used

Manual Review

Recommendation

Include an incrementing nonce in the transfer ID hash to guarantee uniqueness across operations within the same block.

Issue L-8: MondayStockRouter storage gap does not account for its two state variables [RESOLVED]

Source: <https://github.com/sherlock-audit/2026-03-anchored-mar-9th-2026/issues/32>

Summary

MondayStockRouter declares two state variables (`whitelistEnabled` and `whitelist`) but sets `_gap` to 50, the same as if it had no state variables. The gap should be 48.

Vulnerability Detail

MondayStockRouter inherits StockRouterBase (which has its own `_gap[50]`) and adds two storage variables:

<https://github.com/sherlock-audit/2026-03-anchored-mar-9th-2026/blob/main/anchored-contracts/contracts/monday/MondayStockRouter.sol#L14-15>

```
bool public whitelistEnabled; // 1 slot
mapping(address user => bool allowed) public whitelist; // 1 slot
```

The gap is declared as 50:

<https://github.com/sherlock-audit/2026-03-anchored-mar-9th-2026/blob/main/anchored-contracts/contracts/monday/MondayStockRouter.sol#L84>

```
uint256[50] private _gap;
```

For comparison, AncStockRouter attempted to account for its own variables and set `_gap` [49] (though it also miscounted - it has `compliance` + `whitelist` = 2 slots but only subtracted 1).

Impact

A future upgrade developer counting available expansion slots from the gap size would overcount by 2, potentially causing a storage collision if 49 or 50 new variables are added in an upgrade.

Tool Used

Manual Review

Recommendation

Issue L-9: Mismatch between the code and the documentation [RESOLVED]

Source: <https://github.com/sherlock-audit/2026-03-anchored-mar-9th-2026/issues/33>

Summary

There are several places where there's a mismatch between the documentation and the actual behavior of the code itself.

Vulnerability Detail

The `contract-architecture.md` file specifies `MarketOrder` and `LimitOrder` fields without fee snapshot fields. However, the code includes both of the fields:

<https://github.com/sherlock-audit/2026-03-anchored-mar-9th-2026/blob/main/anchored-contracts/contracts/base/StockExchangeBase.sol#L135-140>

```
_marketOrders[orderId] = MarketOrder({
    tokenIndex: tokenIdx,
    amount: amount,
    mintFeeRateSnapshot: _mintFeeRate,
    protocolFeeRateSnapshot: _protocolFeeRate
});
```

In the same file, order ID user bits documented as `16-175`. The actual packing in `_packOpId` places the user at bits 0-159 (bottom 160 bits). Extracting bits 16-175 would return corrupted data:

<https://github.com/sherlock-audit/2026-03-anchored-mar-9th-2026/blob/main/anchored-contracts/contracts/base/StockExchangeBase.sol#L632-648>

```
function _packOpId(uint64 counter, uint16 tokenIdx, address user) internal pure
↳ returns (bytes32) {
    return bytes32(
        (uint256(counter) << 192) | (uint256(tokenIdx) << 176) |
        // reserved 16 bits (175-160)
        uint256(uint160(user))
    );
}

function _unpackOpId(bytes32 opId) internal pure returns (address user, uint64
↳ counter, uint16 tokenIdx) {
    counter = uint64(uint256(opId) >> 192);
```

```

    tokenId = uint16(uint256(opId) >> 176);
    user = address(uint160(uint256(opId)));
}

function _unpackTokenIndex(bytes32 opId) internal pure returns (uint16) {
    return uint16(uint256(opId) >> 176);
}

```

Additionally, the depositId/withdrawalId values are not calculated in accordance with the docs (where they're formatted as keccak256 hash):

<https://github.com/sherlock-audit/2026-03-anchored-mar-9th-2026/blob/main/anchored-contracts/contracts/base/CashierBase.sol#L681-683>

```

function _packOpId(uint64 counter, address token, uint32 timestamp) internal pure
    ↪ returns (bytes32) {
    ↪     return bytes32((uint256(counter) << 192) | (uint256(uint160(token)) << 32) |
    ↪         uint256(timestamp));
    ↪ }

```

As you can see, the actual code at `_packOpId` uses packed bytes32: counter (64) | token (160) | timestamp (32). There's no hashing, no user address and no nonce.

Impact

Mismatch between the code and the specification.

Tool Used

Manual Review

Recommendation

Consider either aligning the code to the documentation or changing the documentation so that it is consistent with code.

Disclaimers

Sherlock does not provide guarantees nor warranties relating to the security of the project.

Usage of all smart contract software is at the respective users' sole risk and is the users' responsibility.