IRONNODE

ironnode.io

FINAL REPORT

FBYT

Solana Program Security Audit

Prepared for Tellus Foundation SA

IronNode Security

24 November 2026 / Version 2.1

IRONNODE

01

SECTION 01

Executive summary

1 Executive Summary

1.1 Introduction

FBYT is an Anchor-based Solana vault program. Investors deposit approved assets for shares, authorized managers trade the portfolio, and investors redeem shares for underlying assets. The program also accounts for management, performance and lending fees, trade-risk limits, and blocked-token Claims.

IronNode reviewed the contract, tested transaction paths locally, and retested the client's fixes. This final report records 24 findings: five Medium, eleven Low and eight Informational. Fifteen are Fixed, two are Risk Accepted and seven are Acknowledged. No finding is classified Critical or High. Finding 5 still describes an issuer delegate's ability to remove a full affected balance; the client accepts that authority.

1.2 Scope and revisions

Item	Reviewed scope
Client	Tellus Foundation SA / FBYT
Auditor	Alp Onaran, IronNode Security
Review period	27 October - 24 November 2026
Repository	FirstBytOrganization/FBYT-Money-Management-App-SmartContract
Original audit source	19a732adedf31f9f4cbaf2ccf789f843fdf3f5ce
Final remediation source	eb9b9e379d2cc15015926baf59b099d1026d55cc
Program scope	programs/fbyt_vault/src/
Original scope size	66 Rust files; 10,054 physical lines; 41 exposed instructions

Findings 1-21 retain code excerpts and adverse-case evidence from the original audit revision. Findings 22-24 arose during remediation retesting; their excerpts show the final fix and are identified as such. The physical line count includes comments and blank lines.

This is a contract-only assessment. Frontend, backend, hosting, operational key custody and standalone audits of Jupiter, Pyth, Token-2022 and Klend are outside scope.

1.3 Findings overview

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	5	11	8

24 **FINDINGS**

15 Fixed / 2 Risk Accepted / 7 Acknowledged

Severity describes the original issue. Status records the client's fix or policy decision at the final review. Fixed means the relevant source path passed focused local checks; it does not mean every live transaction was observed.

No.	Finding	Severity	Status
1	Full withdrawal fails when a frozen asset follows a removed asset	Medium	Fixed
2	An activated transfer hook can block payment of a transferable investor claim	Medium	Fixed
3	A claim can close while a mutable token fee leaves the investor no spendable payout	Medium	Fixed
22	Late deposits can dilute existing investors after an oracle price change	Medium	Fixed
23	A Kamino cToken exit can stop when Klend cannot refresh its reserve	Medium	Fixed
4	The fixed loss window can exceed the daily limit inside a rolling day	Low	Risk Accepted
5	A permanent delegate can remove approved assets from funded vaults	Low	Risk Accepted
6	A wide vault portfolio can prevent investor withdrawals	Low	Fixed
7	Splitting trades below the activity threshold can avoid fees	Low	Fixed
8	Waiting for dormancy can erase accrued management fees	Low	Fixed

No.	Finding	Severity	Status
9	Frequent successful deposits can discard management-fee fractions	Low	Fixed
10	A dormant vault can accept deposits after trading stops	Low	Fixed
11	Paying the final claim can leave an empty asset occupying a registry slot	Low	Fixed
12	A one-unit asset and accrued management fee can revert the final investor exit	Low	Fixed
13	An accepted deposit can transfer tokens while minting no investor shares	Low	Fixed
24	Scaled token dust can be removed using the wrong displayed value	Low	Fixed
14	Blocked assets can prevent a whole vault withdrawal	Informational	Acknowledged
15	A full exit can transfer an indivisible asset to the last shareholder	Informational	Acknowledged
16	Cached vault value can enlarge the daily swap-loss budget	Informational	Acknowledged
17	A traded residual can disappear from investor withdrawals	Informational	Acknowledged
18	Unlisted vault tokens can pass to a later depositor	Informational	Acknowledged
19	Cleanup can burn an uncapped residual token balance	Informational	Acknowledged
20	State cleanup can strand a funded unlisted token account	Informational	Acknowledged
21	Positive price exponents are misvalued and large powers can panic	Informational	Fixed

1.4 Executive remediation summary

The client corrected the exit and Claim paths, deposit pricing, fee accounting, registry cleanup and scaled-token valuation in the reviewed source. Full exits now create blocked-asset Claims across registry positions. Activated transfer hooks can be supplied to investor Claim settlement, and a recipient can require a minimum net payout after transfer fees.

Late deposits now read current prices from snapshot-pinned Pyth accounts. Management-fee fractions carry forward, active-period fees survive dormancy, and every successful swap resets activity for fee purposes. Deposits that would mint no shares, or enter a dormant vault, are rejected.

Investors can receive Kamino cTokens in kind when Klend cannot refresh a reserve. The final projection uses Klend interest and the market reward cap; focused tests compared fees and payouts with and without a refresh. The fixes also use displayed amounts for scaled-token dust and reject unsupported price exponents and arithmetic powers.

Two Low policy risks remain accepted. The swap-loss budget uses a fixed 24-hour window, not a rolling window, and approved Token-2022 assets may retain a permanent delegate. The Low ratings reflect the confirmed product policy; acceptance does not remove either technical behavior. These choices should stay visible in asset approval and investor disclosures. Seven Informational observations remain acknowledged, including the maximum-layout blocked-asset limit and trusted cleanup assumptions.

The client reported migrating the legacy nine-asset vault and passing unsigned full-exit simulations for both investors at 39 and 41 accounts. We confirmed the reported production executable hash, but did not receive those simulation payloads or observe a signed exit. Finding 6 is Fixed for the reviewed account-layout change; this does not certify a completed live transfer.

1.5 Review and test record

The original review traced deposits, redemptions, Claims, reserved balances, fees, share valuation, swaps, Jupiter account forwarding, oracle binding, registry changes, cleanup and administration. Local adverse tests were paired with controls that changed asset order, prices, time, token permissions, fee settings and portfolio width.

Evidence set	Recorded result
Original case register	728 unique cases, including controls, model queries and inconclusive setups
Original formal rules	40 source-derived Z3 queries on the baseline revision
Final full LiteSVM suite	126 passed, 0 failed on the reviewed source/test tree
Final program unit suite	115 passed, 0 failed
Focused repository builder checks	9 passed, 0 failed
Final lending controls	Real Klend fixture; refresh, broken-Scope and reward-cap cases

The 728 register entries are case records, not 728 passing contract tests. The final LiteSVM run used an existing locally built SBF artifact associated with the reviewed source/test tree. That local result does not independently prove byte-for-byte equivalence with the deployed binary.

1.6 Formal analysis

The original audit executed 40 Z3 rules derived from the baseline code and business rules. Fifteen selected mathematical or transition properties had no counterexample, four restated source guards, and 21 produced expected counterexamples to stronger desired properties. Counterexamples were checked against executable paths and policy decisions before classification.

The models are not a machine proof of the Rust or SBF binary. The final remediation was assessed with code review and local regression tests; the original formal rules were not re-proved against the revised source.

Rule	Security property	Result
FV-01	Pro-rata payout never exceeds available raw units	No counterexample in integer model
FV-02	Pro-rata floor remainder is bounded by the share denominator	No counterexample in integer model
FV-03	Accepted deposit satisfies the explicit share and value-loss guards	Guard restatement only
FV-04	Partial withdrawal cannot raise the investor high-water mark	No counterexample in integer model

Rule	Security property	Result
FV-05	High-watermark scaling has the expected floor bound	No counterexample in integer model
FV-06	Single-call management fee shares obey the 99% fraction cap	No counterexample in integer model
FV-07	Fee-share allocation cannot consume more than pending shares	No counterexample in integer model
FV-08	Spending no more than available preserves reserve backing	No counterexample in integer model
FV-09	Gross claim payment preserves unreserved token balance	No counterexample in integer model
FV-10	Withdrawal gross fee legs reconcile without a negative leg	No counterexample in integer model
FV-11	Final investor and pending fee shares partition available units	No counterexample in integer model
FV-12	Accepted trade obeys its fixed-window cached-value guard	Guard restatement only
FV-13	Every rolling 24 hours obeys a 2% loss limit	Counterexample; policy conditional
FV-14	Frequent fee accrual equals one accrual over the same period	Counterexample; fee fractions can be discarded
FV-15	Dormant interaction preserves already-earned active fees	Counterexample; timer can advance with no fee
FV-16	Settling a gross claim always yields positive spendable units	Counterexample; 100% token fee
FV-17	Cached-value cap also enforces a current-value cap	Counterexample; accepted client design
FV-18	Integer share minting gives an exact fraction	Expected counterexample control
FV-19	Partial exit cannot allocate more gross units than available	No counterexample in allocation model
FV-20	Blocked and paid withdrawal legs preserve reserve backing	No counterexample with backed starting reserve
FV-21	A third distinct blocked mint always receives a reserve slot	Counterexample; known two-mint claim cap

Rule	Security property	Result
FV-22	Settling the last claim removes its now-empty registry mint	Counterexample; existing empty-slot finding
FV-23	Dust removal keeps every later positive balance registered	Counterexample; existing residual finding
FV-24	On-chain cleanup rejects a positive unreserved token balance	Counterexample; cleanup policy conditional
FV-25	Performance fee cannot exceed investor gross withdrawal	No counterexample under fee-rate bound
FV-26	Edge slippage acceptance bounds midpoint loss	No counterexample for checked oracle values
FV-27	Marginal trading fee is additive when one trade is split	Counterexample; fee-policy candidate
FV-28	A zero daily-loss setting still caps swap losses	Counterexample; unset deployment setting
FV-29	A zero confidence setting rejects uncertain deposit prices	Counterexample; unset deployment setting
FV-30	A zero holding-period setting enforces a positive wait	Counterexample; unset deployment setting
FV-31	Positive Pyth exponent is valued with the correct sign	Counterexample; unsupported-feed condition
FV-32	Power-of-ten u64 scale exists for every u8 mint decimal	Counterexample; unsupported-mint condition
FV-33	Lowering the admin cap instantly trims legacy registries	Counterexample; legacy registry unchanged
FV-34	Application-admin handoff revokes the previous operator	Counterexample; governance decision
FV-35	Public claim settlement cannot outrun admin reassignment	Counterexample; claim-ownership decision
FV-36	Cleanup guard refuses live shares and outstanding claims	Guard restatement only
FV-37	Holding route can spend tokens predating the swap	Guard restatement only
FV-38	Enabled loss-counter saturation hides an over-cap sum	No counterexample with enabled 500-bps ceiling

Rule	Security property	Result
FV-39	Every accepted initial deposit mints positive shares	Counterexample; signed local zero-share deposit and later payout
FV-40	Approved token backing survives an issuer permanent delegate	Counterexample; funded mainnet positions and signed local delegate test

1.7 Evidence and limits

The client reported production deployment of the reviewed remediation source. On 21 November, IronNode read production ProgramData and confirmed the client-reported executable SHA-256 b509b5ae43e9f4b52b4f76da5cdff08b404ed1145869538cdf49ad29f638c526 and finalized upgrade transaction. We did not independently reproduce the solana-verify build mapping the source commit to that binary.

IronNode submitted no live investor-signed withdrawal, live-funds transfer or real Jupiter route. The original nine-asset withdrawal failure was observed before the layout change. Post-upgrade 39/41-account investor simulations are client-reported; their raw transactions and logs were not supplied for independent replay.

The original audit test suite contains baseline scripts, the register, logs and formal runners. The final source repository contains the additional remediation and lending tests cited in Findings 22-24. Finding-page code and test snippets are selected portions, not standalone programs.

This report describes the tested code and recorded policies as of 24 November 2026. Later source changes, approved assets, oracle configurations or external protocol behavior can change the assessed risk. A security review cannot guarantee the absence of all defects.

IRONNODE

02

SECTION 02

Findings

2 Findings

2.1 Full withdrawal fails when a frozen asset follows a removed asset

MEDIUM FIXED

Original assessment: The failure was reproduced on the compiled local devnet-feature build. The production-feature path and the ability of supported launch assets to enter the same frozen state have not been established.

The full-withdrawal instruction processes assets in registry order. It may remove an emptied asset from the registry and return excess account rent to the investor before it processes the next asset. If a later vault token account is frozen, the instruction must create a Claim for that payout. In a funded three-asset vault, freezing the second asset caused the investor-signed full withdrawal to fail with UnbalancedInstruction after the first asset was paid and removed and before the Claim creation CPI. The confirmed failed transaction left the investor shares, registry entries and checked token balances unchanged.

The failure depends on order. A half-share withdrawal with the later frozen asset succeeded and created a Claim. A full withdrawal succeeded when the first registered asset was frozen, and it also succeeded when the first two assets were frozen. These controls distinguish this behavior from the two-Claim reservation limit and the wide-portfolio account limit.

2.1.1 Code location

programs/fbyt_vault/src/instructions/investor_instruction/withdraw_token_fund.rs:365-375, 433-444

```
record_claim(
    &ra[claim_cursor + offset],
    &investor_info,
    &system_program_info,
    vault_pool_key,
    token_mint.key(),
    *recipient,
    *amount,
    *kind,
    current_unix_timestamp,
)?;
// ...
AssetUtils::check_and_remove_asset_if_empty(
    &mut ctx.accounts.asset_registry,
    token_mint.key(),
    &mut InterfaceAccount::<TokenAccount>::try_from(&ra[base + 3])?,
    token_mint.decimals,
    admin_pool.dust_threshold_usd,
    price,
    reserved_after,
)?;
ctx.accounts
    .asset_registry
    .realloc_to_fit(&mut ctx.accounts.investor, &ctx.accounts.system_program)?;
```

programs/fbyt_vault/src/states/mod.rs:133-142

```
if current_minimum_balance > new_minimum_balance {
    let diff = current_minimum_balance - new_minimum_balance;
    self_account_info.sub_lamports(diff)?;
    payer.add_lamports(diff)?;
} else if current_minimum_balance < new_minimum_balance {
    let diff = new_minimum_balance - current_minimum_balance;
    payer.transfer_sol(&self_account_info, diff, system_program)?;
}

self_account_info.resize(new_data_len)?;
```

2.1.2 Impact

An investor cannot finish a full exit in the tested configuration while a later registered vault asset remains frozen and an earlier asset is removed during the exit. A partial withdrawal may work, but it leaves the position open. No asset theft was demonstrated.

2.1.3 Remediation

Solved: The client changed full-exit claim and rent handling so a frozen asset can become a Claim after an earlier asset is removed. Local tests passed with the frozen asset in each registry position.

2.1.4 Proof of concept

The signed local reproduction froze the second vault token account, simulated a full exit, then submitted the investor-signed transaction. It failed with UnbalancedInstruction and the assertions below confirmed unchanged shares, registry entries and balances.

A half-share exit with the later asset frozen succeeded. Full exits succeeded when the first asset, or the first two assets, were frozen. These controls support an interaction between the earlier rent refund and the later Claim CPI; the runtime balance check was not separately instrumented.

revision-19a732a/tests/zz_loc05_claim_realloc_confirmed.test.ts:370-377

```
expect(JSON.stringify(failedRecord!.meta!.err)).to.include("UnbalancedInstruction");
expect((await program.account.investorPool.fetch(investorPoolAddress)).shares.toString())
  .to.equal(sharesBeforeFailedSend);
expect((await program.account.assetRegistry.fetch(registryAddress)).assetMints.map(
  (mint: PublicKey) => mint.toBase58()
)).to.deep.equal(registryBeforeFailedSend);
expect(await tokenAmount(vaultAtas.get(baseMint.toBase58())!)).to.equal(baseBeforeFailedSend);
expect(await tokenAmount(vaultAtas.get(firstMint.toBase58())!)).to.equal(frozenBeforeFailedSend);
```

2.2 An activated transfer hook can block payment of a transferable investor claim

MEDIUM FIXED

Original assessment: The Medium impact requires a held, approved mint whose authority activates a transfer hook. At the recorded mainnet check, the observed hooks were inactive.

An approved Token-2022 mint may be acquired while its transfer hook is inactive. If the mint authority later activates a hook, FBYT treats the mint as blocked even when that hook permits an ordinary token transfer. Investor withdrawal burns shares and creates a reserved claim for the token. The claim-settlement handler then rejects payment with TransferStillBlocked for as long as the hook is active. The investor cannot provide hook accounts through that handler, and FBYT's token CPI does not forward them. This requires action by the mint's hook authority; a permissionless activation path was not shown.

2.2.1 Code location

programs/fbyt_vault/src/utils/token.rs:89-98, 103-114

```
pub fn is_transfer_blocked(
    mint: &InterfaceAccount<Mint>,
    from: &InterfaceAccount<TokenAccount>,
    to: &InterfaceAccount<TokenAccount>,
) -> Result<bool> {
    if from.is_frozen() || to.is_frozen() {
        return Ok(true);
    }
    Ok(is_mint_paused(mint)? || has_transfer_hook(mint)?)
}
// ...
pub fn has_transfer_hook(mint: &InterfaceAccount<Mint>) -> Result<bool> {
    let account_info = mint.to_account_info();
    if account_info.owner != &Token2022::id() {
        return Ok(false);
    }
    let data = account_info.data.borrow();
    let state = StateWithExtensions::<spl_token_2022::state::Mint>::unpack(&data)?;
    Ok(state
        .get_extension::<TransferHook>()
        .map(|hook| Option::<Pubkey>::from(hook.program_id).is_some())
        .unwrap_or(false))
}
```

programs/fbyt_vault/src/instructions/investor_instruction/claim_withheld.rs:71-78

```
if is_transfer_blocked(
    &ctx.accounts.mint,
    &ctx.accounts.vault_token_account,
    &ctx.accounts.recipient_token_account,
)? {
    return err!(ErrorCode::TransferStillBlocked);
}
```

2.2.2 Impact

A shareholder can lose access to that token through FBYT while the hook remains active, although their reserved claim stays backed. The local control directly transferred the token when the hook account was supplied. After hook activation, FBYT rejected a manager sell and a signed investor withdrawal created a 12,500,000-unit claim that claim_withheld could not pay. Removing the hook allowed one settlement. A read-only mainnet check found 17 approved mints with mutable hook authorities, three held by the observed nine-asset vault, but all their hooks were inactive at that check. No live claim lock or theft was observed.

2.2.3 Remediation

Solved: The client added a hook-aware Claim payout path with the required extra accounts and signer restrictions. Local tests paid a claim after hook activation.

2.2.4 Proof of concept

The signed local test passed with a no-op hook that permits ordinary transfers when its program account is supplied. FBYT rejected the reserved-claim payment while the hook was active. After the authority removed the hook, the same stored claim paid and closed. The excerpt below checks both paths.

```
revision-19a732a/tests/zz_phase11_hook_live.test.ts:1096-1108

await expectRejected(pay, "TransferStillBlocked");
expect(await amount(t.vaultAta)).to.equal(vaultBefore);
expect(await connection.getAccountInfo(claims[0])).to.not.equal(null);

// Removing the hook is a counterexample: the same stored claim pays.
await sendAndConfirmTransaction(connection, new Transaction().add(
  createUpdateTransferHookInstruction(t.mint, owner.publicKey,
    PublicKey.default, [], TOKEN_2022_PROGRAM_ID)
), [owner]);
const investorBeforePay = await amount(investorAta);
await pay();
expect(await amount(investorAta) - investorBeforePay).to.equal(grossClaim);
expect(await connection.getAccountInfo(claims[0])).to.equal(null);
```

2.3 A claim can close while a mutable token fee leaves the investor no spendable payout

MEDIUM FIXED

Original assessment: The Medium impact requires a supported mint whose issuer can raise the transfer fee materially after a claim is created. The launch policy for these mints is not confirmed.

An investor can leave while the recipient account is frozen, creating a gross claim that reserves vault tokens. If the issuer later thaws the account and raises a Token-2022 transfer fee, any signer can settle the claim. The program transfers the gross amount and closes the claim without checking the recipient's net spendable receipt.

The issuer must control the token fee and the frozen account. An outsider cannot change the fee. The local production-feature test used a synthetic mint and compared the same claim at 1% and 100% transfer fees.

2.3.1 Code location

programs/fbyt_vault/src/instructions/investor_instruction/claim_withheld.rs:13-14, 19-33, 79-91

```
/// Anyone may send this: the tokens can only go to the claim's recipient.
pub payer: Signer<'info>,
// ...
#[account(
    mut,
    has_one = vault_pool,
    has_one = mint,
    has_one = rent_payer,
    seeds = [
        CLAIM_SEED.as_bytes(),
        vault_pool.key().as_ref(),
        mint.key().as_ref(),
        claim.recipient.as_ref(),
        &[claim.kind],
    ],
    bump = claim.bump,
    close = rent_payer,
)]
// ...
let amount = ctx.accounts.claim.amount;
transfer_to_user(
    false,
    &ctx.accounts.vault_pool,
    &ctx.accounts.vault_token_account,
    &ctx.accounts.recipient_token_account,
    &ctx.accounts.mint,
    &ctx.accounts.token_program,
    &ctx.accounts.token_program2022,
    amount,
)?;

ctx.accounts.vault_pool.release_reserved(mint_key, amount)?;
```

2.3.2 Impact

In the 100% fee branch, an unrelated signer closed a 99-unit claim. The recipient received zero spendable units and the 99 units appeared as withheld fees. At 1%, the same claim paid 98.01 spendable units. A direct withdrawal also netted zero at 100%, so the FBYT-specific exposure is forced settlement of an existing claim without a fresh investor decision. No live mint or claimant loss was tested.

2.3.3 Remediation

Solved: Claim settlement now needs the recipient's signature when a transfer fee applies and checks the recipient's chosen minimum net balance increase. A failed minimum leaves the Claim open.

2.3.4 Proof of concept

The production-feature local SBF test passed 1/1. It also checked claim backing, failed payment while frozen, atomic rollback, and replay rejection. The issuer authority and clock epochs were synthetic.

```
revision-19a732a/tests/zz_phase20_full_fee_claim.test.ts:349-355

const old = await execute(config.olderTransferFee.epoch);
const changed = await execute(config.newerTransferFee.epoch);
expect(changed.settled.all[1].amount).to.equal(0n);
expect(changed.settled.all[2].amount).to.equal(0n);
expect(changed.settled.all[1].withheld).to.equal(99n * UNIT);
expect(changed.settled.all[2].withheld).to.equal(99n * UNIT);
expect(changed.settled.reserved).to.equal(0n);
```

2.22 Late deposits can dilute existing investors after an oracle price change

MEDIUM FIXED

The stale-snapshot deposit path was reproduced locally on the earlier remediation revision. The current deposit path and its pinned-price controls passed focused LiteSVM tests.

A late deposit used the vault's recent PriceSnapshot to value existing holdings. The snapshot could remain valid for 60 seconds after a newer price had appeared on chain. A depositor could therefore subscribe against an old portfolio value and receive too many shares when the holdings had risen in price.

In the local reproduction, a $100 deposit after the price move minted 100,000,001 shares and later redeemed for $126.66. Refreshing the snapshot first minted 66,666,667 shares and redeemed for $99.99. The current code instead reads each pinned oracle account during the deposit.

2.22.1 Code location

The code excerpt shows the final remediation in eb9b9e3; the adverse result was measured on the earlier remediation source.

eb9b9e3/programs/fbyt_vault/src/instructions/investor_instruction/deposit_token_fund.rs:202-224

```
        // prices on chain now. The snapshot only names each asset's Pyth
        // account (and whether it is a cToken); every price is read from that
        // account here, under the deposit's age and confidence limits.
        let vault_pool_value_usd = {
            let snapshot = ctx.accounts.price_snapshot.load()?;
            let ledger = ctx.accounts.vault_ledger.load()?;
            snapshot.require_pins(&asset_registry.asset_mints, current_unix_timestamp as i64,
                ctx.accounts.admin_pool.oracle_config_version)?;
            let kamino = KaminoHoldings::load(
                ra,
                asset_registry.asset_mints.len() * HOLDING_ACCOUNTS_PER_ASSET,
                &snapshot_flags(&snapshot, asset_registry.asset_mints.len()),
                &asset_registry.asset_mints,
                vault_pool_key,
                ctx.accounts.admin_pool.key(),
                crate::utils::Accrual::ThisSlot,
            )?;
            let prices_start = asset_registry.asset_mints.len() * HOLDING_ACCOUNTS_PER_ASSET +
                kamino.accounts_len;
            require!(ra.len() >= prices_start + asset_registry.asset_mints.len(),
                ErrorCode::InvalidAccountData);
            let clock = Clock::get()?;
            vault_holdings_usd(ra, &asset_registry.asset_mints, vault_pool_key, &ledger, &kamino,
                |i| {
                get_checked_price_at(&ra[prices_start + i], snapshot.deposit_price_account(i)?,
                &clock, price_max_age, max_price_conf_bps).map(Some)
            })?
        };
```

2.22.2 Impact

A depositor could acquire value belonging to existing shareholders when a supported asset's on-chain price rose after the snapshot was written. The observed value transfer was local; no live-funds deposit was submitted.

2.22.3 Remediation

Solved: A late deposit now prices every holding from its pinned canonical Pyth account at deposit time, subject to age and confidence limits. The snapshot names the accounts but no longer supplies stale prices for share minting.

2.22.4 Proof of concept

The focused local test moved one holding from $1 to $2 after the snapshot, then required a $100 deposit to mint shares against the $3,000 current portfolio rather than the $2,000 stored value. Wrong price-account and missing-account controls were rejected.

eb9b9e3/tests-litesvm/tests/audit_remediation.rs:1114-1123

```
/// The auditor's case: the snapshot still holds SOLX at $1, the chain says $2.
/// The deposit is priced against the vault at $3,000, not $2,000, so it buys
/// no part of the price move from the existing investors.
#[test]
fn n1_a_late_deposit_mints_shares_at_the_current_prices() {
    let mut vw = vault_after_a_price_move();
    let outstanding = vault_pool(&vw).outstanding_shares().unwrap();
    deposit_result(&mut vw, 21, 100_000_000).unwrap();
    let expected = (100_000_000u128 * outstanding as u128 / 3_000_000_000u128) as u64;
    assert_eq!(shares_of(&vw, 21), expected, "$100 against $3,000 (stored prices: $2,000)");
```

2.23 A Kamino cToken exit can stop when Klend cannot refresh its reserve

MEDIUM FIXED

The original failure was reproduced locally against the real Klend binary with an unusable Scope account. The final in-kind exit and fee projection passed the tested broken-oracle, refresh-omitted and reward-cap cases.

The first lending withdrawal path required a same-slot Klend reserve refresh even when FBYT only needed to transfer cTokens to an investor. If Klend could not refresh because the reserve's Scope oracle account was unusable, the investor could not withdraw the cTokens in kind.

The client removed that dependency for in-kind exits and added an internal reserve projection. A follow-up retest found that fee results could differ without a refresh; the final revision models Klend interest and the lending-market reward cap for the tested cases.

2.23.1 Code location

The code excerpt shows the final projection in eb9b9e3; the original failed exit was reproduced on the earlier remediation source.

eb9b9e3/programs/fbyt_vault/src/utils/kamino_holdings.rs:124-156

```
        let rewards_max_apr_bps = if market_len == 1 {
            crate::klend::market_rewards_max_apr_bps(&ra[start + 1], &config.market)?
        } else {
            0
        };

        let clock = Clock::get()?;
        let mut holdings = Vec::with_capacity(flagged.len());
        for (n, &index) in flagged.iter().enumerate() {
            let position_info = &ra[start + 1 + market_len + 2 * n];
            let reserve_info = &ra[start + 2 + market_len + 2 * n];
            let position = Account::<KaminoPosition>::try_from(position_info)?;
            let expected_position = Pubkey::create_program_address(
                &[
                    KAMINO_POSITION_SEED.as_bytes(),
                    vault_pool_key.as_ref(),
                    position.reserve.as_ref(),
                    &[position.bump],
                ],
                &crate::ID,
            )
            .map_err(|_| error!(ErrorCode::InvalidKaminoPositionAccounts))?;
            require!(
                position_info.key() == expected_position
                    && position.vault_pool == vault_pool_key
                    && position.collateral_mint == registry[index]
                    && reserve_info.key() == position.reserve,
                ErrorCode::InvalidKaminoPositionAccounts
            );
            let reserve = match accrual {
                Accrual::ThisSlot => ReserveState::load_fresh(reserve_info, &clock)?,
                Accrual::ProjectedToNow =>
                ReserveState::load(reserve_info)?.accrued_to(clock.slot, clock.unix_timestamp,
                rewards_max_apr_bps)?,
            };
```

2.23.2 Impact

An external reserve-oracle failure could prevent investor withdrawal of otherwise transferable cTokens. A mismatched reserve projection could also alter the investor payout or FBYT lending fee. The tested final path continued the in-kind exit without a refresh and matched the refreshed fee and payout.

2.23.3 Remediation

Solved: In-kind Kamino exits can proceed without a Klend refresh. FBYT projects reserve interest and capped rewards with Klend's rules, preserving the tested investor payout and lending fee when Scope cannot refresh.

2.23.4 Proof of concept

The local test made Klend refresh fail by replacing the Scope account, then compared full and half withdrawals with refresh, without refresh, and with broken Scope. The returned lending fee, investor cTokens and remaining position matched in the tested cases.

eb9b9e3/tests-litesvm/tests/kamino_lending.rs:925-932

```
#[test]
fn n2_the_lending_fee_does_not_depend_on_the_refresh() {
    for part in [2, 1] {
        let refreshed = exit_after_a_month(Exit::Refreshed, part);
        assert!(refreshed.0 > 0, "a month of interest earns a lending fee");
        assert_eq!(exit_after_a_month(Exit::RefreshOmitted, part), refreshed, "refresh left out
            (part 1/{part})");
        assert_eq!(exit_after_a_month(Exit::OracleBroken, part), refreshed, "klend cannot refresh
            (part 1/{part})");
    }
```

2.4 The fixed loss window can exceed the daily limit inside a rolling day

LOW RISK ACCEPTED

Final rating: Low under the client's confirmed fixed-window policy. The original local boundary test and its impact remain documented below.

The vault records swap loss in a 24-hour bucket that starts with a swap. At the bucket boundary it clears the previous loss before adding the next swap. An authorized manager can use most of one bucket's allowance just before the boundary and use a fresh allowance one second later. Individual trades still pass the slippage check.

The local test refreshed oracle data and vault value before the second sequence. The refreshed value was lower, so stale pricing did not cause the acceptance. This case is separate from the earlier cached-value denominator observation.

2.4.1 Code location

programs/fbyt_vault/src/states/vault_pool.rs:396-415

```
pub fn record_swap_loss(&mut self, loss_usd: u64, limit_bps: u16, now: u64) -> Result<()> {
    if now >= self.loss_window_start.saturating_add(LOSS_WINDOW_SECONDS) {
        self.loss_window_start = now;
        self.loss_window_usd = 0;
    }
    self.loss_window_usd = self.loss_window_usd.saturating_add(loss_usd);
    if limit_bps > 0 {
        require!(
            self.value_usd > 0 && now <= self.value_at.saturating_add(LOSS_WINDOW_SECONDS),
            ErrorCode::VaultValueUnknown
        );
        require!(
            self.loss_window_usd as u128 * 10_000
                <= self.value_usd as u128 * limit_bps as u128,
            ErrorCode::DailyLossLimitReached
        );
    }
    Ok(())
}
```

2.4.2 Impact

With a 100-bps per-trade slippage setting and a 2% bucket cap, the manager-signed local swaps incurred 3.07% of initial value across the one-second boundary. The same attempted swap before reset failed atomically. The route used a deterministic Jupiter stand-in; no real route or live loss was demonstrated. The client confirmed that this fixed-window behavior is the intended policy.

2.4.3 Remediation

Risk Accepted: The client retained the first-trade-anchored 24-hour loss window as the intended rule and does not promise a rolling-day limit.

2.4.4 Proof of concept

The production-feature local swap test passed 1/1. A separate 500-bps control passed 1/1 and measured 3.87% across the same boundary.

```
revision-19a732a/tests/zz_phase21_live_style_loss_window.test.ts:251-267, 279-280

expect(beforeControl.loss * 10_000n <= initialValue * 200n).to.equal(true);
expect(beforeControl.base).to.equal(972_240_000_000n);
expect(beforeControl.quote).to.equal(12_240_000_000n);
const thirdIx = await swapIx(false, 960n * UNIT, 952_320_000_000n);
const sameWindow = await run(thirdIx, manager);
expect(sameWindow.ok).to.equal(false);
expect(errorOf(sameWindow.logs)).to.equal("DailyLossLimitReached");
expect(await snapshot()).to.deep.equal(beforeControl);

const boundary = t0 + DAY;
await setTime(boundary);
const refreshedBoundary = await run(await refreshIx(), manager);
expect(refreshedBoundary.ok, refreshedBoundary.logs).to.equal(true);
const valueAtBoundary = (await snapshot()).value;
expect(valueAtBoundary < initialValue).to.equal(true);
const third = await run(thirdIx, manager);
expect(third.ok, third.logs).to.equal(true);
// ...
const combined = beforeControl.loss + after.loss;
expect(combined * 10_000n > initialValue * 280n).to.equal(true);
```

2.5 A permanent delegate can remove approved assets from funded vaults

LOW RISK ACCEPTED

Final rating: Low under the client's explicit acceptance of issuer permanent-delegate authority. A delegate can still remove a full affected balance; no mainnet seizure was observed.

FBYT approves a token by setting its OraclePool approval flag without checking the Token-2022 PermanentDelegate extension. A permanent delegate can move or burn that token from a vault account without the vault signing. The token balance can fall while the investor's shares and FBYT's asset registry remain unchanged. Only the configured delegate, or someone who controls its key, can use this authority.

A finalized mainnet check on 7 November found 16 approved mints with nonzero permanent delegates. Twelve were present in 27 funded token accounts across 16 FBYT vaults, all with outstanding shares. The legacy nine-asset vault held three such mints and its known external investor still held shares. PYUSD was approved but was not in any registered vault. The signed local test used a synthetic mint; it did not exercise any issuer authority on mainnet.

2.5.1 Code location

programs/fbyt_vault/src/instructions/oracle_instruction/oracle_approve.rs:38-48

```
pub fn handle_approve_oracle_pool(ctx: Context<ApproveOraclePool>) -> Result<()> {
    let oracle_pool = &mut ctx.accounts.oracle_pool;

    oracle_pool.oracle_approve()?;

    emit!(ApproveOraclePoolEvent {
        admin_pool: ctx.accounts.admin_pool.key(),
        token_mint: ctx.accounts.token_mint.key(),
    });
    Ok(())
}
```

programs/fbyt_vault/src/states/oracle.rs:71-74

```
pub fn oracle_approve(&mut self) -> Result<()> {
    self.is_approved = true;
    Ok(())
}
```

2.5.2 Impact

A token issuer's delegate can remove the entire balance of an affected mint from a funded vault. Investors may retain shares with no corresponding token backing. The local test removed ten synthetic tokens and a later full exit paid zero of that mint. The 27 funded mainnet positions establish present exposure, not a mainnet drain, realized loss, or permissionless attack. Their USD value was not calculated in this check.

2.5.3 Remediation

Risk Accepted: The client continues to allow approved Token-2022 mints with permanent delegates, accepting the issuer's ability to move or burn affected vault balances.

2.5.4 Proof of concept

The local test tests/zz_audit_permanent_delegate_probe.test.ts approved a synthetic Token-2022 mint, funded a vault with ten tokens, rejected an outsider's transfer, and let the configured delegate move all ten. Investor shares remained; a later signed full exit paid zero. Finalized mainnet reads independently checked the approved mint extensions, vault registries, token-account ownership, nonzero balances and outstanding shares. The deployed ELF has not been matched to the pinned source.

```
revision-19a732a/tests/zz_audit_permanent_delegate_probe.test.ts:113-115, 123-126, 141-142

const delegatedTransfer = await move(issuerDelegate);
expect((await connection.confirmTransaction(delegatedTransfer, "confirmed"))
  .value.err).to.equal(null);
// ...
expect(vaultAfter).to.equal(0n);
expect((await balance(thiefAta)).amount).to.equal(10n * UNIT);
expect((await program.account.investorPool.fetch(pool)).shares.eq(sharesBefore))
  .to.equal(true);
// ...
expect(await connection.getAccountInfo(pool)).to.equal(null);
expect((await balance(investorAta)).amount).to.equal(0n);
```

2.6 A wide vault portfolio can prevent investor withdrawals

LOW FIXED

Original assessment: Rated Low for the existing nine-asset vault under the client's seven-asset launch policy. The prepared investor exit exceeds the tested 64-account limit; an investor-signed mainnet attempt has not been submitted.

withdraw_token_fund requires seven account entries for every mint in the vault's asset registry. It processes the whole registry in one transaction, even when an investor withdraws only part of their shares. The admin can increase the permitted asset count without checking whether a withdrawal at that count fits within transaction limits.

An authorized manager can build a portfolio of approved assets through the normal swap flow. In the local test, both half-share and full-share withdrawals from a 20-asset vault failed before the contract executed. A partial withdrawal did not reduce the accounts required.

2.6.1 Code location

programs/fbyt_vault/src/instructions/investor_instruction/withdraw_token_fund.rs:71-73, 144-153

```
/// Number of remaining_accounts per asset: OraclePool, PriceUpdateV2, Mint, From, To,
    MoneyManager, Admin.
/// Claim accounts for blocked legs follow the 7 x N block (see the second pass).
const ACCOUNTS_PER_ASSET: usize = 7;
// ...
let ra = ctx.remaining_accounts;
// Snapshot: the dust check below can remove a mint from the registry
// mid-loop, and the remaining_accounts layout follows the registry as it
// was when the transaction was built.
let asset_mints: Vec<Pubkey> = ctx.accounts.asset_registry.asset_mints.clone();
let asset_count = asset_mints.len();
require!(
    ra.len() >= asset_count * ACCOUNTS_PER_ASSET,
    ErrorCode::InvalidAccountData
);
```

programs/fbyt_vault/src/states/admin_pool.rs:300-303

```
pub fn admin_pool_max_asset_count_update(&mut self, max_asset_count: u16) -> Result<()> {
    self.max_asset_count = max_asset_count;
    Ok(())
}
```

2.6.2 Impact

Investors in an affected vault cannot complete the normal partial or full withdrawal until an authorized manager reduces the portfolio, or the contract is changed. This is a denial of access to vault assets, not demonstrated theft. The recorded admin cap was seven, but one existing vault holds nine funded registered assets and has an external shareholder. A normal withdrawal account list for that investor compiles to 73 unique keys, or 74 with a compute-budget instruction, above the tested 64-key limit. In the 64-account local replay, seven assets passed and eight distinct-feed-equivalent assets failed with TooManyAccountLocks. Manager consolidation to seven assets is planned but has not been verified.

2.6.3 Remediation

Solved: The client reduced withdrawal accounts per asset and migrated the legacy nine-asset vault to the ledger and snapshot layout. Its two full-exit simulations were reported at 39 and 41 accounts; their raw RPC responses and a signed exit were not independently reviewed.

2.6.4 Proof of concept

The original 20-asset versioned withdrawals used an address lookup table and a 1,400,000-unit compute budget. Half and full exits exceeded the local validator's then-enabled 128-account limit. That result was followed by a separate replay with the 64-account feature setting.

In the focused replay, seven distinct-feed-equivalent assets used 60 keys and passed; eight used 67 keys and failed with TooManyAccountLocks before program execution. A later signed seven-asset full exit closed the investor account and paid all transferable assets. The recorded nine-asset mainnet account list required 73 keys, or 74 with a compute instruction. Its unsigned simulation was rejected before execution; no investor-signed mainnet withdrawal was submitted.

```
archive/early-work/zz_loc04_mainnet_limit.test.ts:355-360, 366-367

if (ASSET_COUNT === 7) {
  expect(ordinary.error, ordinary.logs).to.equal(null);
} else if (ASSET_COUNT === 8) {
  expect(JSON.stringify(ordinary.error)).to.include("TooManyAccountLocks");
  expect(ordinary.units ?? 0).to.equal(0);
}
// ...
expect((await program.account.investorPool.fetch(investorPoolAddress)).shares.toString())
  .to.equal(pool.shares.toString());
```

2.7 Splitting trades below the activity threshold can avoid fees

LOW FIXED

Original assessment: The fee bypass was reproduced on a production-feature local build. It requires trading authority and nonzero fees.

The swap guard updates last_trade_at only when one trade sells at least the configured percentage of cached vault value. Withdrawal and management-fee paths use that timestamp to decide whether the vault has traded at all. An authorized trader can therefore make several smaller, profitable swaps while leaving last_trade_at at its creation value. The vault traded and earned profit, but the fee paths treat it as untraded.

In the local test, one 4.5-unit swap and five 0.9-unit swaps left identical 95.5-base and 4.525-quote balances. At the observed production activity threshold of 1%, the large-trade vault paid 249,999 raw units of performance fee on exit; the split-trade vault paid zero. A separate management-fee test paid 27,007 raw units in the large-trade branch and zero in the split branch. At a permitted 0.5% threshold, both branches paid the same performance fee. The adverse trader needs trading authority; the manager may have little incentive to waive its own fee.

2.7.1 Code location

programs/fbyt_vault/src/instructions/swap/guard.rs:103-105

```
if vault_pool.is_activity(input_usd, admin_pool.activity_min_bps) {
    vault_pool.vault_update_last_trade_at()?;
}
```

programs/fbyt_vault/src/states/vault_pool.rs:419-421, 463-469

```
pub fn is_activity(&self, input_usd: u64, activity_min_bps: u16) -> bool {
    input_usd as u128 * 10_000 >= self.value_usd as u128 * activity_min_bps as u128
}
// ...
pub fn mm_fee_applies(&self, idle_period: u64, now: u64) -> bool {
    self.total_shares > 0
        && self.last_trade_at > self.created_at
        && now <= self.get_idle_period_end(idle_period)
        && self.raised_amount_usd >= self.min_raise_amount_usd
        && !(self.vault_pool_status == VAULT_CLOSED && self.closed_at == 0)
}
```

programs/fbyt_vault/src/instructions/investor_instruction/withdraw_token_fund.rs:127, 278-287

```
need_get_performance_fee = vault_pool.last_trade_at > vault_pool.created_at;
// ...
let success_fee: u64 = if need_get_performance_fee {
    calc_performance_fee(
        withdraw_amount,
        investor_vault_pool_value_usd,
        investor_pool.hight_watermark,
        vault_pool.performance_fee,
```

```
programs/fbyt_vault/src/instructions/investor_instruction/withdraw_token_fund.rs:127, 278-287 (continued)

    )?
} else {
    0u64
};
```

2.7.2 Impact

Where nonzero fees are intended for trading activity, an authorized trader can reduce the fee recipients' allocations by splitting otherwise equivalent trades. Investors retain the avoided amounts. The local proof measures the accounting difference. It does not measure a large live loss or give outsiders trading access. It used a locally built devnet-feature SBF and deterministic Jupiter stand-in; focused tests also replayed the 19a732a production-feature build on 2 November 2026.

2.7.3 Remediation

Solved: Every successful swap now updates trading activity, so splitting trades below the old activity threshold no longer avoids the fee trigger.

2.7.4 Proof of concept

Four focused local suites passed 1/1 each, including the production swap instruction with a deterministic mock route and a lower-threshold control. The decisive assertions from tests/zz_phase1_jupiter_activity_fee_bypass.test.ts are:

The full test also asserts the two vaults finish with identical base and quote token balances before withdrawal.

```
revision-19a732a/tests/zz_phase1_jupiter_activity_fee_bypass.test.ts:154-158, 230-237

const largeState = await program.account.vaultPool.fetch(large.vault);
const splitState = await program.account.vaultPool.fetch(split.vault);
expect(BigInt(largeState.lastTradeAt.toString()) > large.createdAt).to.equal(true);
expect(BigInt(splitState.lastTradeAt.toString())).to.equal(split.createdAt);
expect(largeState.totalShares.toString()).to.equal(splitState.totalShares.toString());
// ...
const splitPaid = await withdraw(split);
const largePaid = await withdraw(large);
const total = (DEPOSIT - LARGE_IN) + LARGE_OUT;
expect(splitPaid[0] + splitPaid[1]).to.equal(total);
expect(splitPaid.slice(2).every(x => x === 0n)).to.equal(true);
const fee = largePaid.slice(2).reduce((a, b) => a + b, 0n);
expect(fee > 0n).to.equal(true);
expect(largePaid[0] + largePaid[1] + fee).to.equal(total);
```

2.8 Waiting for dormancy can erase accrued management fees

LOW FIXED

Original assessment: The client confirms that active-period fees remain payable after dormancy or closure. Signed local production-feature tests reproduce their loss.

The vault checks whether management fees apply at the time of a deposit or withdrawal. Once the idle deadline has passed, it records zero fee shares for the entire period since the last accrual, including the time when the vault was still active. It then advances the fee timer. An investor who waits until dormancy can leave that earned fee with investors instead of paying the manager and protocol. A deposit into an open-ended dormant vault has the same effect. Closing just before dormancy does not preserve the unrecorded fee.

2.8.1 Code location

```
programs/fbyt_vault/src/states/vault_pool.rs:463-469, 550-560

pub fn mm_fee_applies(&self, idle_period: u64, now: u64) -> bool {
    self.total_shares > 0
        && self.last_trade_at > self.created_at
        && now <= self.get_idle_period_end(idle_period)
        && self.raised_amount_usd >= self.min_raise_amount_usd
        && !(self.vault_pool_status == VAULT_CLOSED && self.closed_at == 0)
}
// ...
let added = if applies {
    self.mm_fee_shares_due(current_unix_timestamp)?
} else {
    0
};
self.pending_mm_fee_shares = self
    .pending_mm_fee_shares
    .checked_add(added)
    .ok_or(ErrorCode::CalculateOverflow)?;
self.last_mm_fee_withdraw_at = current_unix_timestamp;
Ok(added)
```

2.8.2 Impact

Fee recipients can lose management fees earned during the active part of the interval. In the signed local test, an exit one second before dormancy paid 27,396,250 raw fee-token units; an otherwise identical exit one second after paid zero. This was a synthetic 1,000-unit portfolio at a 1% annual fee and a one-day idle period. The recorded mainnet configuration used a 90-day idle period and a seven-day collection interval, so the test does not establish the same loss on a live vault. Investor principal theft was not shown.

2.8.3 Remediation

Solved: Management fees earned while the vault was active are accrued before dormancy stops new fees. The owed amount remains collectible.

2.8.4 Proof of concept

revision-19a732a/tests/zz_phase14_dormant_fee_boundary.test.ts; passing log: evidence/revision-19a732a-zz_phase14_dormant_fee_boundary.log (1 passing, 0 failing). The test used signed FBYT transactions and controlled local time.

Controls showed that an intermediate accrual preserved its fee, an admin idle extension restored fee eligibility, and a late operator collection was rejected with VaultIsDormant. The client confirms that fees earned before dormancy or close must remain payable.

```
revision-19a732a/tests/zz_phase14_dormant_fee_boundary.test.ts:263-268, 277-282

const activeFeeBefore = await beforeIdle.feeBalance();
const activeExit = await beforeIdle.withdraw();
expect(activeExit.ok, activeExit.logs).to.equal(true);
const activeFee = (await beforeIdle.feeBalance()) - activeFeeBefore;
expect(activeFee > 0n).to.equal(true);
expect(big((await beforeIdle.vaultState()).totalShares)).to.equal(0n);
// ...
const dormantFeeBefore = await afterIdle.feeBalance();
const dormantExit = await afterIdle.withdraw();
expect(dormantExit.ok, dormantExit.logs).to.equal(true);
const dormantFee = (await afterIdle.feeBalance()) - dormantFeeBefore;
expect(dormantFee).to.equal(0n);
expect(big((await afterIdle.vaultState()).totalShares)).to.equal(0n);
```

2.9 Frequent successful deposits can discard management-fee fractions

LOW FIXED

Original assessment: The client confirms fee fractions must carry forward. The compiled local reproduction used the devnet feature; production-feature execution and a material supported-vault scenario remain unverified.

In an open-ended traded vault, a permitted investor deposit accrues management fees before new shares are minted. The fee calculation rounds the newly owed shares down to an integer. The accrual routine then advances the fee timer even if that integer is zero. A series of successful deposits can therefore split an otherwise payable interval into slices that each round to zero, with no remainder carried forward.

In our local test, an investor signed 143 deposits ten minutes apart over 23 hours and 50 minutes. Every deposit succeeded and advanced the timer, while pending management-fee shares stayed at zero. An independent calculation using only the original one million shares gives a floor of 27 fee-share units for that interval; a delayed-deposit control accrued 27. The two branches have different later capital paths, so the 27-unit comparison is used only with the original-share lower bound.

2.9.1 Code location

programs/fbyt_vault/src/states/vault_pool.rs:526-539, 550-560

```
pub fn mm_fee_shares_due(&self, current_unix_timestamp: u64) -> Result<u64> {
    let end = self.mm_fee_end(current_unix_timestamp);
    if end <= self.last_mm_fee_withdraw_at || self.total_shares == 0 {
        return Ok(0);
    }
    let elapsed = (end - self.last_mm_fee_withdraw_at) as u128;
    let year_bps = 10_000u128 * WITHDRAW_PERIOD_YEAR as u128;
    let fee = (self.money_management_yearly_fee as u128 * elapsed).min(year_bps * 99 / 100);
    let shares = (self.outstanding_shares()? as u128)
        .checked_mul(fee)
        .ok_or(ErrorCode::CalculateOverflow)?
        / (year_bps - fee);
    u64::try_from(shares).map_err(|_| ErrorCode::CalculateOverflow.into())
}
// ...
let added = if applies {
    self.mm_fee_shares_due(current_unix_timestamp)?
} else {
    0
};
self.pending_mm_fee_shares = self
    .pending_mm_fee_shares
    .checked_add(added)
    .ok_or(ErrorCode::CalculateOverflow)?;
self.last_mm_fee_withdraw_at = current_unix_timestamp;
Ok(added)
```

2.9.2 Impact

The manager and protocol can collect fewer management-fee shares than the elapsed time and original share base would otherwise earn. In this fixture, 27 shares were worth about $0.062 at the initial share value. Investor principal was not lost. Actual transaction costs, approved assets and a plausible higher-value scenario have not been established; this test does not support a High or Medium rating.

2.9.3 Remediation

Solved: Fractional management-fee shares are carried across deposits and later accruals instead of being discarded at each calculation.

2.9.4 Proof of concept

The executed local test tests/zz_phase3_fee_fragmentation.test.ts used a Bankrun fork of the same funded vault snapshot and asserted success and zero pending fees after every investor-signed deposit. Its decisive loop and independent bound were:

The test passed. The client confirms the fractional entitlement must carry forward and notes that swaps also invoke fee accrual. A production-feature rerun and an economically material supported-vault example remain to be completed.

archive/early-work/zz_phase3_fee_fragmentation.test.ts:191-203, 224-226

```
for (let i = 1; i <= depositCount; i++) {
  await fragmented.setTime(t0 + BigInt(i * 600));
  const before = await fragmented.state();
  const result = await fragmented.deposit();
  expect(result.ok, result.logs).to.equal(true);
  const after = await fragmented.state();
  if (i === 1 || i === depositCount) console.log("P3 fee fragmented deposit", i,
    "totalShares", after.totalShares.toString(),
    "pending", after.pendingMmFeeShares.toString(),
    "timer", after.lastMmFeeWithdrawAt.toString());
  expect(BigInt(after.totalShares.toString()) >
      BigInt(before.totalShares.toString())).to.equal(true);
  expect(after.pendingMmFeeShares.toString()).to.equal("0");
  expect(after.lastMmFeeWithdrawAt.toString()).to.equal((t0 + BigInt(i * 600)).toString());
// ...
expect(BigInt(onceState.pendingMmFeeShares.toString()) > 0n).to.equal(true);
expect(originalSharesFeeFloor >= 27n).to.equal(true);
expect(fragmentedState.pendingMmFeeShares.toString()).to.equal("0");
```

2.10 A dormant vault can accept deposits after trading stops

LOW FIXED

Original assessment: The client confirms dormant subscriptions are unintended. Signed local production-feature tests recovered the deposit after the holding period.

An open-ended vault remains marked active after its idle deadline. Its deposit path checks that stored status but does not apply the idle deadline. Trading and manager closure do apply the deadline, so an investor can successfully deposit into a vault that the manager can no longer trade or close through the normal instructions. The new deposit can also restart the holding period for that investor's entire position.

On the locally built client-revision production-feature program, a signed deposit after the deadline succeeded, a manager-signed close returned VaultIsDormant, an immediate investor exit returned HoldingPeriodNotEnded, and the exit succeeded after the configured 900-second hold. A separate local test confirmed that a manager-signed trade was rejected after a dormant deposit. The tested investor withdrew after the 900-second hold. No principal was lost.

2.10.1 Code location

programs/fbyt_vault/src/instructions/investor_instruction/deposit_token_fund.rs:136-141, 160-161

```
// Check if the vault is in the correct state to accept deposits
if vault_pool.vault_pool_status != VAULT_ACTIVE {
    return err!(ErrorCode::VaultNotActive);
}

// Check if the deposit amount is within the allowed range
// ...
} else if vault_pool.is_open_ended {
    let ra = ctx.remaining_accounts;
```

programs/fbyt_vault/src/instructions/money_manager_instruction/close_vault.rs:45-49

```
if current_unix_timestamp > idle_period_end {
    return err!(ErrorCode::VaultIsDormant);
}
```

2.10.2 Impact

An investor can add capital to a nontrading vault and wait up to the configured holding period before withdrawing. This is an on-chain availability and disclosure concern if subscriptions to dormant vaults are not intended. The test recovered the deposited funds after 900 seconds. It did not test a live frontend restriction, a deployed binary, or real assets. Admin changes to the idle period may offer a privileged recovery path, but that path was not exercised in this case.

2.10.3 Remediation

Solved: Late deposits now check the idle deadline and reject a dormant vault before taking investor funds.

2.10.4 Proof of concept

tests/zz_phase7_holding_reset.test.ts passed against the exact locally built 19a732a production-feature SBF. It used signed transactions and a copied Pyth account in the local test bank. The relevant assertions are:

The full test also asserts that manager closure fails with VaultIsDormant, an immediate exit fails with HoldingPeriodNotEnded, and the investor exits successfully after 900 seconds.

```
revision-19a732a/tests/zz_phase7_holding_reset.test.ts:238-255

await setTime(dormantAt);
const dormantDeposit = await run(await depositIx(first, firstPool,
  firstAta, EXTRA), first);
expect(dormantDeposit.ok, dormantDeposit.logs).to.equal(true);
expect(BigInt((await poolState(firstPool)).lastDepositAt.toString()))
  .to.equal(dormantAt);
const managerClose = await run(await program.methods.closeVault()
  .accountsPartial({ adminPool, vaultPool: vault,
    moneyManager: manager.publicKey, tokenMint: mint })
  .instruction(), manager);
expect(errorOf(managerClose.logs)).to.equal("VaultIsDormant");
const dormantEarlyExit = await run(await withdrawIx(first, firstPool,
  firstAta, firstOldShares), first);
expect(errorOf(dormantEarlyExit.logs)).to.equal("HoldingPeriodNotEnded");
await setTime(dormantAt + HOLD);
const dormantMatureExit = await run(await withdrawIx(first, firstPool,
  firstAta, firstOldShares), first);
expect(dormantMatureExit.ok, dormantMatureExit.logs).to.equal(true);
```

2.11 Paying the final claim can leave an empty asset occupying a registry slot

LOW FIXED

Original assessment: Reproduced at the seven-asset cap. A subsequent partial withdrawal cleared the empty registry slot in the tested state.

A claimed mint remains in the vault's asset registry while its reserved tokens are present. When the final claim is paid, the claim instruction transfers the reserved tokens and releases the reserve, but does not update the registry. The now-empty mint can occupy the last permitted slot and block a manager's swap into a different approved asset.

This state requires a blocked payout, a manager sale of the remaining unreserved balance, and later settlement of the claim. The freeze authority and manager both participated in the executed fixture. The empty slot blocked a new-output swap until a partial exit cleared it. No investor loss was observed.

2.11.1 Code location

programs/fbyt_vault/src/instructions/investor_instruction/claim_withheld.rs:79-91

```
let amount = ctx.accounts.claim.amount;
transfer_to_user(
    false,
    &ctx.accounts.vault_pool,
    &ctx.accounts.vault_token_account,
    &ctx.accounts.recipient_token_account,
    &ctx.accounts.mint,
    &ctx.accounts.token_program,
    &ctx.accounts.token_program2022,
    amount,
)?;

ctx.accounts.vault_pool.release_reserved(mint_key, amount)?;
```

programs/fbyt_vault/src/states/asset_registry.rs:53-67

```
pub fn add_asset_mint(&mut self, mint: Pubkey, max_asset_count: u16) -> Result<()> {
    // Check if mint already exists
    if self.asset_mints.contains(&mint) {
        return Ok(()); // Already exists, no need to add
    }

    // Add the new mint
    self.asset_mints.push(mint);

    // Check if the number of assets exceeds the max asset count
    if self.asset_mints.len() > max_asset_count as usize {
        return err!(ErrorCode::MaxAssetCountExceeded);
    }
    Ok(())
}
```

2.11.2 Impact

The seven-asset local vault's next new-output swap failed with MaxAssetCountExceeded after the claimed mint's vault balance and reserve both reached zero. A second investor's partial withdrawal removed the empty entry, after which the same swap succeeded. This does not establish a permanent withdrawal lock or a live deployment effect.

2.11.3 Remediation

Solved: Paying the final Claim now removes the empty mint from the asset registry, freeing its slot for later activity.

2.11.4 Proof of concept

The production-feature local seven-asset test passed 1/1 using a versioned transaction and lookup table. It checked reserve backing, token supply, share conservation, and rollback on the failed swap.

```
revision-19a732a/tests/zz_phase16_empty_claim_asset_slot.test.ts:311-319, 329-338

expect(await amount(ata(quote, vault))).to.equal(0n);
expect(await reserveOfQuote()).to.equal(0n);
expect((await program.account.assetRegistry.fetch(registry)).assetMints
  .some((mint) => mint.equals(quote))).to.equal(true);

const beforeFailedSwap = await Promise.all(mints.flatMap((mint) => [
  ata(mint, vault), ata(mint, poolPda),
]).map(amount));
await expectFailure(() => trade(base, third, 10n * UNIT, 10n * UNIT),
// ...
await assertLedger("second partial exit removes empty mint");
const afterRemoval = (await program.account.assetRegistry.fetch(registry)).assetMints;
expect(afterRemoval.length).to.equal(6);
expect(afterRemoval.some((mint) => mint.equals(quote))).to.equal(false);
expect(afterRemoval.some((mint) => mint.equals(base))).to.equal(true);
await trade(base, third, 10n * UNIT, 10n * UNIT);
await assertLedger("new-asset trade after removal");
const finalMints = (await program.account.assetRegistry.fetch(registry)).assetMints;
expect(finalMints.length).to.equal(7);
expect(finalMints.some((mint) => mint.equals(third))).to.equal(true);
```

2.12 A one-unit asset and accrued management fee can revert the final investor exit

LOW FIXED

Original assessment: Signed production-feature local tests reproduce this failure with a valuable zero-decimal unit and with a sub-cent raw unit of a six-decimal token. A split exit recovered the divisible base funds in the six-decimal case.

On a final investor exit, one remaining raw asset unit and positive accrued management-fee shares make the investor's pro-rata leg round to zero. The current rule gives the last unit to the fee leg, then attempts a zero-amount investor transfer. The helper returns ZeroTransfer and reverts every asset payout. A signed production-feature test reproduced this with a six-decimal quote asset whose remaining raw unit was worth about $0.0023 and a fresh price. A tiny output-side swap relisted that unit without a dust check. The vault still held almost 99 divisible base tokens when its full exit failed.

2.12.1 Code location

programs/fbyt_vault/src/instructions/investor_instruction/withdraw_token_fund.rs:262-276, 384-395

```
let available = vault_pool.available(&token_mint.key(), from_account.amount);
let withdraw_amount: u64 = pro_rata_amount(available, shares, outstanding_shares)?;

// The owed management fee leaving with these shares, in this asset.
// The last shares out take the exact remainder, so rounding the two
// parts separately cannot leave units behind in an empty vault.
let money_management_fee: u64 = if shares == vault_pool.total_shares {
    available
        .checked_sub(withdraw_amount)
        .ok_or(ProgramError::ArithmeticOverflow)?
} else {
    // Floored, like the periodic charge: rounding favours those who stay.
    u64::try_from(available as u128 * owed_fee_shares as u128 / outstanding_shares as u128)
        .map_err(|_| ProgramError::ArithmeticOverflow)?
};
// ...
if withdraw_amount > 0 || money_management_fee > 0 {
    if !investor_blocked {
        transfer_to_user(
            true,
            vault_pool,
            &from_account,
            &to_account,
            &token_mint,
            &ctx.accounts.token_program,
            &ctx.accounts.token_program2022,
            investor_amount,
        )?;
```

programs/fbyt_vault/src/utils/token.rs:49-55

```
if amount == 0 {
    if revert_on_zero {
        return err!(ErrorCode::ZeroTransfer);
    } else {
        return Ok(());
```

programs/fbyt_vault/src/utils/token.rs:49-55 (continued)

```
    }
}
```

2.12.2 Impact

The fee-bearing investor's one-step full exit reverted atomically while a zero-fee twin exited. In the six-decimal closed-vault control, a partial exit followed by a final exit recovered the almost 99 base tokens but left the sub-cent quote unit unlisted in the vault. An active manager could also swap the unit away and restore full exit. The valuable zero-decimal test left the last shares blocked after a partial exit, but its mint and route were synthetic. The tests establish a full-exit failure. They do not establish a permanent lock of launch assets or a live loss.

2.12.3 Remediation

Solved: Final-exit allocation now gives the last indivisible unit to the investor while preserving the fee ledger's accounting.

2.12.4 Proof of concept

revision-19a732a/tests/zz_phase12_zero_investor_leg.test.ts and revision-19a732a/tests/zz_phase18_6decimal_dust_exit.test.ts passed on locally built production-feature SBF with signed FBYT transactions and deterministic Jupiter stand-ins. The six-decimal log is evidence/revision-19a732a-zz_phase18_6decimal_dust_exit.log (1 passing).

The fresh-priced six-decimal case failed full exit with ZeroTransfer; the zero-fee twin passed. The closed-vault split-exit control recovered the divisible base balance and left one quote raw unit behind. A stale price within the configured discontinuation grace returns PriceNeedsRefresh earlier in the instruction; that particular stale-price path was not the basis of this finding.

revision-19a732a/tests/zz_phase18_6decimal_dust_exit.test.ts:224-234, 279-283

```
expect((await program.account.investorPool.fetch(feeVault.pool)).shares
  .eq(sharesBefore)).to.equal(true);
expect(await amount(feeVault.baseAta)).to.equal(feeBeforeBase);
expect(await amount(feeVault.quoteAta)).to.equal(1n);

const noFeeShares = (await program.account.investorPool.fetch(noFeeVault.pool)).shares;
const quoteInvestorBefore = await amount(ata(quoteMint, investor.publicKey));
await exit(noFeeVault, noFeeShares);
expect(await amount(ata(quoteMint, investor.publicKey)) - quoteInvestorBefore)
  .to.equal(1n);
expect(await amount(noFeeVault.quoteAta)).to.equal(0n);
// ...
expect(quoteListedAfterPartial).to.equal(false);
const closedRest = (await program.account.investorPool.fetch(closedVault.pool)).shares;
await exit(closedVault, closedRest);
expect(await connection.getAccountInfo(closedVault.pool)).to.equal(null);
expect(await amount(closedVault.quoteAta)).to.equal(1n);
```

2.13 An accepted deposit can transfer tokens while minting no investor shares

LOW FIXED

Original assessment: A signed local test confirmed the loss of contributed units for zero shares. Material exposure with supported asset prices and minimum contributions remains unverified.

The deposit handler checks the contribution's oracle value, then floors the token amount into six-decimal shares. During the raise, and when an open-ended vault has no outstanding shares, it does not reject a zero result. It updates investor and vault accounting and transfers the tokens even though the depositor receives no shares. A later shareholder can receive those tokens on exit.

In the signed local sequence, an investor deposited 999 raw units of a nine-decimal token and received zero shares. A second investor deposited 1,000 raw units, received one share, then withdrew all 1,999 units. The empty open-ended vault reproduced the behavior after the raise. A default $0.01 minimum also allowed it when the synthetic unit price was $100,000; an ordinary minimum-contribution control rejected the small deposit. No material loss with a launch-approved asset was demonstrated.

2.13.1 Code location

programs/fbyt_vault/src/instructions/investor_instruction/deposit_token_fund.rs:150-159, 201-214, 221-233

```
let shares = if current_unix_timestamp < raise_end {
    vault_value_after = vault_pool
        .value_usd
        .checked_add(deposit_amount_usd)
        .ok_or(ErrorCode::CalculateOverflow)?;
    parse_to_decimals(
        amount_without_fee,
        ctx.accounts.token_mint.decimals,
        SHARE_DECIMALS,
    )?
// ...
if vault_pool.total_shares != 0 {
    // Shares outstanding over a vault worth nothing: a 1:1 mint would
    // hand part of this deposit to the existing holders.
    require!(vault_pool_value_usd != 0, ErrorCode::VaultHasNoValue);
    shares_for_deposit(
        vault_pool.outstanding_shares()?,
        deposit_amount_usd,
        vault_pool_value_usd,
    )?
} else {
    parse_to_decimals(
        amount_without_fee,
        ctx.accounts.token_mint.decimals,
        SHARE_DECIMALS,
// ...
vault_pool.vault_increase_investor_count_if_needed(investor_pool.shares)?;
investor_pool.investor_deposit(shares, deposit_amount_usd)?;
vault_pool.vault_deposit_fund(deposit_amount_usd, shares)?;
vault_pool.set_value(vault_value_after, current_unix_timestamp);

transfer_to_vault(
    &ctx.accounts.investor,
    &ctx.accounts.from_account,
```

```
programs/fbyt_vault/src/instructions/investor_instruction/deposit_token_fund.rs:150-159, 201-214, 221-233
(continued)

    &ctx.accounts.to_account,
    &ctx.accounts.token_mint,
    &ctx.accounts.token_program,
    amount,
)?;
```

2.13.2 Impact

A depositor can lose the contributed token units while retaining no redemption claim. The later shareholder received the first investor's 999 raw units in the local test. For a nine-decimal token, each affected deposit is below one six-decimal share unit; the USD amount depends on the token price and configured minimum. The $100,000 price was synthetic, not a claim about a supported mint.

2.13.3 Remediation

Solved: The deposit path now rejects a zero-share result before the token transfer can complete.

2.13.4 Proof of concept

The signed local tests tests/zz_audit_raise_zero_share_probe.test.ts and tests/zz_audit_raise_default_min_high_price.test.ts checked share counts, token balances, later shareholder exit and a rejecting contribution-minimum control. The second test used a synthetic $100,000 unit price. Formal rule FV-39 returned a counterexample to the desired positive-share property; the signed test is the reachability evidence.

```
revision-19a732a/tests/zz_audit_raise_zero_share_probe.test.ts:94-96, 124-126

expect(firstAfterLater.shares.toNumber()).to.equal(0);
expect(laterAfterDeposit.shares.toNumber()).to.equal(1);
expect(afterLater.amount).to.equal(1999n);
// ...
expect(laterBalance.amount).to.equal(1999n);
expect(firstFinal.shares.toNumber()).to.equal(0);
expect((await getAccount(connection, vaultAta)).amount).to.equal(0n);
```

2.24 Scaled token dust can be removed using the wrong displayed value

LOW FIXED

The boundary behavior was tested locally with an SPYx-style ScaledUiAmount mint. The final code values the displayed balance on the tested dust paths.

A Token-2022 ScaledUiAmount mint displays more units than its raw token balance. The original dust checks valued the raw balance, so a balance worth more than the configured dust threshold by displayed amount could be classified as dust and removed from the registry.

The retest used the mint's displayed-unit multiplier at the $0.01 boundary. A 1,318-unit raw remainder stayed listed at about $0.010005 displayed value, while 1,317 units were treated as dust at about $0.009998.

2.24.1 Code location

The code excerpt shows the corrected displayed-amount valuation in eb9b9e3.

eb9b9e3/programs/fbyt_vault/src/utils/util.rs:325-339

```
        if vault_token_account.key() != expected || vault_token_account.mint != *mint ||
            vault_token_account.owner != vault_pool_key {
            return err!(ErrorCode::InvalidFromAccount);
        }
        let available = ledger.available(mint, vault_token_account.amount);
        let (amount, decimals) = match kamino.get(i) {
            Some(h) => {
                h.ctokens(available)?;
                h.net_underlying()?
            }
            None => (
                crate::utils::ui_scaled_amount(&token_mint.to_account_info(), available,
                Clock::get()?.unix_timestamp)?,
                token_mint.decimals,
            ),
        };
        if let Some(price) = price_of(i)? {
```

2.24.2 Impact

A supported scaled token could leave the withdrawal registry while still exceeding the configured economic dust threshold. The reproduced boundary difference was small; a material investor loss was not established.

2.24.3 Remediation

Solved: The dust checks now value ScaledUiAmount tokens by displayed amount on swaps, withdrawals, fee claims and withheld claims. The 1,317/1,318 raw-unit boundary tests passed.

2.24.4 Proof of concept

The focused LiteSVM test sold SPYx down to 1,318 and 1,317 raw units. It asserted opposite registry outcomes at the displayed-value threshold, and companion tests exercised fee and withdrawal paths.

```
eb9b9e3/tests-litesvm/tests/audit_remediation.rs:1567-1575

fn an_xstock_remainder_is_dust_by_its_displayed_value() {
    // 1,318 raw × 1.007474126658 = 1,327 displayed → $0.010005 (raw: $0.009937)
    assert!(spyx_listed_after_selling_down_to(1_318), "above the threshold: stays listed");
    // 1,317 raw → 1,326 displayed → $0.009998
    assert!(!spyx_listed_after_selling_down_to(1_317), "under the threshold: dust");
}

/// `claim_vault_fees` for SPYx (nothing owed, so only its dust check runs);
/// whether SPYx is still in the registry afterwards.
```

2.14 Blocked assets can prevent a whole vault withdrawal

INFORMATIONAL ACKNOWLEDGED

Original assessment: Recorded under the client's accepted two-reservation policy. The separate seven-asset, two-blocked account-limit failure is also documented.

The vault has two reservation slots for blocked asset mints. Three different frozen vault source accounts exhaust those slots and make the whole withdrawal fail. Separately, at the seven-asset launch cap, adding the Claim accounts needed for two blocked mints produced 66 distinct transaction accounts in the local replay and failed before contract execution. One blocked mint used 60 accounts and passed. These are separate limits.

This requires freeze authority over three token accounts in a manager-created portfolio. The test did not show that an ordinary outsider can impose these freezes.

2.14.1 Code location

programs/fbyt_vault/src/states/vault_pool.rs:77, 268-276

```
pub reserved: [ReservedSlot; 2],
// ...
    let slot = self
        .reserved
        .iter_mut()
        .find(|slot| slot.mint == Pubkey::default())
        .ok_or(ErrorCode::TooManyFrozenAssets)?;
    slot.mint = mint;
    slot.amount = amount;
    Ok(())
}
```

programs/fbyt_vault/src/instructions/investor_instruction/withdraw_token_fund.rs:473-475

```
for (mint, amount) in new_reserved {
    ctx.accounts.vault_pool.add_reserved(mint, amount)?;
}
```

2.14.2 Impact

In the four-asset local case, a half-share withdrawal failed with three frozen vault source accounts; after one thawed, an otherwise equivalent withdrawal passed. At seven assets, the two-blocked-asset attempt failed with TooManyAccountLocks at zero compute units. Neither test demonstrated theft. The freezes require token-issuer authority, and no investor-signed mainnet exit was submitted.

2.14.3 Remediation

Acknowledged: The client documented the remaining account limit. At the 12-asset, three-Kamino-position maximum, five blocked legs fit and six can exceed 64 accounts; a staged exit was not added.

2.14.4 Proof of concept

LOC02-local-poc.test.ts:229-238 froze three vault source accounts:

The signed half-share withdrawal simulation reached the program and returned TooManyFrozenAssets; LOC02-local-poc.test.ts:397-401 checked that specific error:

The expected failure was simulated rather than submitted as a confirmed failed transaction. The two-frozen-account control was confirmed on the local validator.

```
Recorded local reproduction

for (const mint of [baseMint, blockedB, blockedC]) {
  await freezeAccount(
    connection,
    issuer,
    vaultAtas.get(mint.toBase58())!,
    mint,
    issuer,
    [],
    { commitment: "confirmed" }
  );
}
// ...
const failed = await withdraw(true);
expect(failed.simulation!.value.err).to.not.equal(null);
const failureLogs = (failed.simulation!.value.logs ?? []).join(" ");
expect(failureLogs).to.include("TooManyFrozenAssets");
```

Selected archived excerpt; the original standalone harness is not included in the package.

2.15 A full exit can transfer an indivisible asset to the last shareholder

INFORMATIONAL ACKNOWLEDGED

Original assessment: Future asset-approval risk. The recorded survey found 55 approved mints with five to nine decimals and no approved valuable indivisible unit.

In an open-ended vault, an investor's withdrawal is calculated separately for every asset and rounded down to that asset's smallest unit. If the vault holds one valuable zero-decimal token and more than one investor has shares, the first investor to exit can receive none of that token. The transaction still removes all their shares. The remaining investor receives the token on the final exit.

This path requires an admin-approved mint and price feed, a manager trade that puts the token in the vault, another investor with shares, and the affected investor's own signed withdrawal. A later investor cannot force someone else to exit. The exit instruction has no minimum total payout check that would reject a material shortfall.

Controls confirmed that a sole investor received the whole token. When the smaller later investor exited first, that investor received no quote token and the original investor received it on the final exit. The value shift follows exit order; the later investor benefits only when another holder exits first.

2.15.1 Code location

programs/fbyt_vault/src/utils/util.rs:443-447

```
pub fn pro_rata_amount(amount: u64, shares: u64, outstanding_shares: u64) -> Result<u64> {
    let part = (amount as u128 * shares as u128)
        .checked_div(outstanding_shares as u128)
        .ok_or(ErrorCode::CalculateOverflow)?;
    u64::try_from(part).map_err(|_| ErrorCode::CalculateOverflow.into())
```

programs/fbyt_vault/src/instructions/investor_instruction/withdraw_token_fund.rs:262-263, 486-489

```
let available = vault_pool.available(&token_mint.key(), from_account.amount);
let withdraw_amount: u64 = pro_rata_amount(available, shares, outstanding_shares)?;
// ...
let (investor_prev_amount, investor_updated_amount) =
    investor_pool.investor_withdraw(shares)?;

let (vault_prev_amount, vault_updated_amount) = vault_pool.vault_withdraw_fund(shares)?;
```

2.15.2 Impact

In the concentrated local case, the first investor's fair oracle-priced claim was 1.000000990 token-equivalents. Their full signed exit paid 0.009900999 base tokens and no quote token. The remaining investor received the whole quote token and a small base remainder. The first investor's shortfall was 0.990099991 token-equivalents, or 99.01% of their fair claim, and the remaining investor received the same excess.

The local test shows a rounding transfer between shareholders. It did not measure a market loss. The zero-decimal token was created for the test and mapped to the captured ETH/USD feed. Launch applicability depends on the approved mint list, the real token's unit value and liquidity, and a production Jupiter route. No deployed program or live funds were tested.

2.15.3 Remediation

Acknowledged: The client treats valuable indivisible units as an asset-approval decision. No approved launch asset with this profile was identified in the audit survey.

2.15.4 Proof of concept

The original local test traded a nine-decimal base asset into one synthetic zero-decimal token and admitted a second investor. A signed full exit paid the first investor no quote token; the final investor received it.

The scenario and divisible-asset, sole-holder and reversed-order controls were replayed on the locally compiled production-feature program with a deterministic Jupiter stand-in. Selected assertions follow. A real route and a supported valuable indivisible mint were not established.

```
revision-19a732a/tests/zz_phase12_indivisible_production.test.ts:286-292, 301-302, 314, 319

expect(abs(BigInt(control.victimShortfall)) <= 2n).to.equal(true);
expect(abs(BigInt(control.lastExcess)) <= 2n).to.equal(true);
const concentratedResult = results[2];
expect(concentratedResult.quoteDecimals).to.equal(0);
expect(BigInt(concentratedResult.victimShortfall) > 950_000_000n).to.equal(true);
expect(BigInt(concentratedResult.victimPaid) < 20_000_000n).to.equal(true);
expect(BigInt(concentratedResult.lastExcess) > 950_000_000n).to.equal(true);
// ...
expect(soloQuotePaid).to.equal(1n);
expect(await balance(solo.vaultQuote)).to.equal(0n);
// ...
expect(reverseLastQuotePaid).to.equal(0n);
// ...
expect(reverseVictimQuotePaid).to.equal(1n);
```

2.16 Cached vault value can enlarge the daily swap-loss budget

INFORMATIONAL ACKNOWLEDGED

Original assessment: The client accepts the cached-value basis of this circuit breaker. It does not enforce 2% of the current vault value.

The swap guard prices each trade with current Pyth prices and records its loss. The daily limit then compares cumulative losses against vault_pool.value_usd, a stored value set by a deposit, withdrawal, or explicit refresh. A swap accepts that value for up to 24 hours without repricing the portfolio. If asset prices fall during that period, an authorized trader can complete swaps whose cumulative loss is above 2% of current vault value while remaining below 2% of the older stored value.

2.16.1 Code location

programs/fbyt_vault/src/instructions/swap/guard.rs:89-102

```
let input_usd = calc_usd_amount(input_amount, input_decimals, input_price)?;
let output_usd = calc_usd_amount(output_amount, output_decimals, output_price)?;
check_slippage_at_edges(
    input_usd,
    &input_price,
    output_usd,
    &output_price,
    swap_max_slippage_bps(admin_pool.max_slippage_bps, input_oracle_pool, output_oracle_pool),
)?;
vault_pool.record_swap_loss(
    input_usd.saturating_sub(output_usd),
    admin_pool.daily_loss_limit_bps,
    now,
)?;
```

programs/fbyt_vault/src/states/vault_pool.rs:401-412

```
self.loss_window_usd = self.loss_window_usd.saturating_add(loss_usd);
if limit_bps > 0 {
    require!(
        self.value_usd > 0 && now <= self.value_at.saturating_add(LOSS_WINDOW_SECONDS),
        ErrorCode::VaultValueUnknown
    );
    require!(
        self.loss_window_usd as u128 * 10_000
            <= self.value_usd as u128 * limit_bps as u128,
        ErrorCode::DailyLossLimitReached
    );
}
```

2.16.2 Impact

With a simulated 5% asset-price decline over six hours, the deployment guide's 2% daily limit and 60-second swap price age, and the creation script's 0.5% slippage cap, six manager-signed swaps recorded $4,574.54 of loss. That exceeded 2% of current vault value ($4,394.98) but remained below 2% of the cached value ($4,626.29). The sixth swap failed when the same vault was refreshed first. An outsider's swap failed authorization. This weakens a control on an authorized trader; the test does not show an outsider acquiring trade authority.

2.16.3 Remediation

Acknowledged: The client confirmed that the swap-loss budget intentionally uses the cached vault value, including the permitted cache age.

2.16.4 Proof of concept

The local release-feature contract was tested with a deterministic mock Jupiter route. The PoC changes a captured Pyth fixture's price and timestamp inside a local bank, then compares identical manager-signed swap sequences with and without a vault-value refresh. Excerpt from NAV01-stale-nav-release-settings-poc.test.ts:260-280:

The recorded exact-settings local run passed; three additional 5% price-fall runs reproduced the same figures. These runs used synthetic oracle updates and a mock route. They do not establish live price history, real liquidity or deployed-source equivalence.

```
Recorded local reproduction

const freshBank = await start([], accounts);
await setFallenPrice(freshBank, now);
const refreshed = await run(freshBank, refreshIx, manager);
expect(refreshed.ok, refreshed.logs).to.equal(true);
const currentNav = BigInt((await vaultState(freshBank, vault)).valueUsd.toString());
const freshSeries = await runSeries(freshBank);
expect(freshSeries.successes).to.equal(5);
expect(freshSeries.failure).to.include("DailyLossLimitReached");

const staleBank = await start([], accounts);
await setFallenPrice(staleBank, now);
const outsiderIx = await makeSwapIx(outsider.publicKey, true, DEPOSIT);
const outsiderTry = await run(staleBank, outsiderIx, outsider);
expect(outsiderTry.logs).to.include("UnauthorizedTrader");
const staleSeries = await runSeries(staleBank);
expect(staleSeries.successes, staleSeries.failure).to.equal(6);
const loss = BigInt((await vaultState(staleBank, vault)).lossWindowUsd.toString());
expect(loss > (currentNav * 200n) / 10_000n).to.equal(true);
```

Selected archived excerpt; the original standalone harness is not included in the package.

2.17 A traded residual can disappear from investor withdrawals

INFORMATIONAL ACKNOWLEDGED

Original assessment: Acknowledged under the client policy for trade-created balances below the dust threshold.

An authorized trader can leave a small, nonzero balance of an approved asset in a vault after a normal swap. When the balance is worth less than the configured dust threshold, the swap or a later withdrawal removes that mint from the asset registry without transferring or recording the remaining tokens. Later withdrawals build their asset list from the registry, so they omit that balance even if it remains in the vault account.

Two otherwise identical local vaults held the same 1,000 raw units after a manager-signed trade. With a zero dust threshold, the asset stayed listed and the final shareholder received the residue. With the $0.01 threshold observed in the admin settings, the mint was removed, both investor exits paid none of that asset, and the 1,000 units remained after shares reached zero. The residual was approximately $0.0023 at removal. A later price increase and material loss were not demonstrated.

2.17.1 Code location

programs/fbyt_vault/src/utils/asset_utils.rs:66-78

```
// Without a price the value is unknown, so only an empty balance is dust.
let is_dust = match price {
    Some(price) => {
        calc_usd_amount(token_ata_account.amount, token_decimals, price)?
            < dust_threshold_usd
    }
    None => token_ata_account.amount == 0,
};

// If balance is less than dust threshold, remove the asset from registry
if is_dust {
    asset_registry.remove_asset_mint(mint)?;
```

programs/fbyt_vault/src/instructions/swap/jupitor.rs:515-525

```
AssetUtils::check_and_remove_asset_if_empty(
    &mut ctx.accounts.asset_registry,
    ctx.accounts.input_mint.key(),
    &mut ctx.accounts.vault_input_token_account,
    ctx.accounts.input_mint.decimals,
    ctx.accounts.admin_pool.dust_threshold_usd,
    Some(input_price),
    ctx.accounts
        .vault_pool
        .reserved_of(&ctx.accounts.input_mint.key()),
)?;
```

2.17.2 Impact

Shareholders can lose the normal on-chain route to a traded residual. The demonstrated amount is below one cent; it does not support a material-loss or theft claim. If the asset later appreciates, it remains absent from the registry and normal redemption. The local test used synthetic assets, a copied Pyth fixture and a deterministic Jupiter stand-in, not a live route or deposited client funds. The relevant code is unchanged in the client revision, but its production-feature binary was not used for this PoC.

2.17.3 Remediation

Acknowledged: The client accepts the dust policy for small trade-created balances after a mint leaves the registry.

2.17.4 Proof of concept

The executed local test tests/zz_phase3_jupiter_dust_delisting.test.ts passed 1/1. Its decisive control and assertions are:

The excerpt omits the first investor's exit; the test performs it in both branches before the last investor exits. The saved run checks token balances and registry state after each transition.

archive/early-work/zz_phase3_jupiter_dust_delisting.test.ts:217-219, 235-240

```
const controlLastQuote = await amount(lastQuote) - lastQuoteBeforeControl;
expect(controlLastQuote).to.equal(controlResidual);
expect(await amount(control.vaultQuote)).to.equal(0n);
// ...
await withdraw(dust, last, dust.lastPool);
const dustLastQuote = await amount(lastQuote) - lastQuoteBeforeDust;
expect(dustLastQuote).to.equal(0n);
expect(await amount(dust.vaultQuote)).to.equal(dustResidual);
expect((await program.account.vaultPool.fetch(dust.vault)).totalShares.toString())
  .to.equal("0");
```

2.18 Unlisted vault tokens can pass to a later depositor

INFORMATIONAL ACKNOWLEDGED

Original assessment: The client treats the demonstrated sender-funded direct credits as unsolicited. Entitlement to later-appreciating vault dust was not established.

The vault removes a mint from its asset registry when its remaining balance is below the dust threshold, even if the token account still holds tokens. Withdrawals iterate the registry, so they omit that unlisted account. A direct transfer to it can then remain in the vault while the last shareholder exits. If a new investor subsequently deposits into an open-ended vault, the mint is registered again and that investor can withdraw the earlier balance.

The proof of concept used a transfer signed by the donor. It does not show that an attacker can force someone else to send tokens.

2.18.1 Code location

programs/fbyt_vault/src/utils/asset_utils.rs:64-78

```
token_ata_account.reload()?;

// Without a price the value is unknown, so only an empty balance is dust.
let is_dust = match price {
    Some(price) => {
        calc_usd_amount(token_ata_account.amount, token_decimals, price)?
            < dust_threshold_usd
    }
    None => token_ata_account.amount == 0,
};

// If balance is less than dust threshold, remove the asset from registry
if is_dust {
    asset_registry.remove_asset_mint(mint)?;
```

programs/fbyt_vault/src/instructions/investor_instruction/withdraw_token_fund.rs:144-149

```
let ra = ctx.remaining_accounts;
// Snapshot: the dust check below can remove a mint from the registry
// mid-loop, and the remaining_accounts layout follows the registry as it
// was when the transaction was built.
let asset_mints: Vec<Pubkey> = ctx.accounts.asset_registry.asset_mints.clone();
let asset_count = asset_mints.len();
```

programs/fbyt_vault/src/instructions/investor_instruction/deposit_token_fund.rs:195-198

```
asset_registry.add_asset_mint(
    ctx.accounts.token_mint.key(),
    ctx.accounts.admin_pool.max_asset_count,
)?;
```

2.18.2 Impact

If direct credits to a vault account belong to its current shareholders, those shareholders can lose their entitlement when the last share exits. A later depositor can receive the balance instead. The test demonstrates misallocation of a donor-supplied balance, not an arbitrary drain of listed vault holdings. The focused local suite passed 10 of 10 assertions, including a control where a listed asset was paid out correctly.

2.18.3 Remediation

Acknowledged: The client classifies unsolicited direct token credits as outside the existing shareholders' entitlement; the resulting later-depositor treatment is an accepted policy.

2.18.4 Proof of concept

Excerpt from H01-local-poc.test.ts:359-372 showing the donor-signed transfer:

The final shareholder then received none of that mint while the account retained the balance. The later depositor's exit asserted receipt of the earlier balance in H01-local-poc.test.ts:393-407:

```
Recorded local reproduction

const gift = 100n * TOKEN;
const giftSignature = await web3.sendAndConfirmTransaction(
  connection,
  new web3.Transaction().add(
    createTransferInstruction(
      ata(mint, victim.publicKey),
      vaultAta,
      victim.publicKey,
      gift
    )
  ),
  [owner, victim],
  confirmOptions
);
// ...
const unlistedBalance = await balance(vaultAta);
expect(unlistedBalance > 100n * TOKEN).to.equal(true);
const captorBefore = await balance(ata(mint, investor.publicKey));
const seedDeposit = 1_000_000n;
const depositSignature = await deposit(investor, noValueVault, seedDeposit);
expect(await registryMints(noValueVault)).to.deep.equal([mint.toBase58()]);
const pool = await program.account.investorPool.fetch(await investorPoolOf(investor,
    noValueVault));
expect(big(pool.shares) > 0n).to.equal(true);
const exitSignature = await withdraw(investor, attacker, noValueVault, pool.shares);
const captorAfter = await balance(ata(mint, investor.publicKey));
expect(captorAfter - captorBefore).to.equal(unlistedBalance);
expect(await balance(vaultAta)).to.equal(0n);
```

Selected archived excerpt; the original standalone harness is not included in the package.

2.19 Cleanup can burn an uncapped residual token balance

INFORMATIONAL ACKNOWLEDGED

Original assessment: Recorded under the accepted zero-share residual policy and trusted-operator model. The backend dust check is outside this audit.

After all investor shares and reserved claims are gone, an authorized operator can reclaim a vault-owned token account. For a non-native token, the instruction burns its entire balance. The instruction does not check the balance against the configured dust threshold. A source comment says the backend refuses balances above that threshold, but that backend was not part of the reviewed code.

2.19.1 Code location

programs/fbyt_vault/src/instructions/rent_reclaim/reclaim_vault_token_accounts.rs:100-121

```
// Wrapped SOL cannot be burned (NativeNotSupported); closing the
// account unwraps it, so the dust goes to the manager with the rent.
let burned = if token_account.is_native() {
    0
} else {
    token_account.amount
};
if burned > 0 {
    token_interface::burn_checked(
        CpiContext::new_with_signer(
            token_program_info.clone(),
            token_interface::BurnChecked {
                mint: mint_info.clone(),
                from: token_account_info.clone(),
                authority: vault_pool.to_account_info(),
            },
            signer_seeds,
        ),
        burned,
        mint.decimals,
    )?;
}
```

programs/fbyt_vault/src/states/vault_pool.rs:433-439

```
pub fn is_rent_reclaimable(&self, idle_period: u64, now: u64) -> bool {
    // Reserved tokens belong to open claims; closing the ATA would burn them.
    self.total_shares == 0
        && !self.has_reserved()
        && (self.vault_pool_status == VAULT_CLOSED
            || now > self.get_idle_period_end(idle_period))
}
```

2.19.2 Impact

An operator transaction can irreversibly burn a residual balance of any size permitted by the token account, if an off-chain precheck is absent, incorrect, or bypassed. The local test burned 10,000,000,000 token base units despite a configured dust threshold. Its synthetic mint and price fixture do not establish a real-world dollar loss. The test did not burn assets belonging to an active shareholder.

2.19.3 Remediation

Acknowledged: The client relies on its trusted cleanup operator and backend dust check before burning a zero-share residual. The backend was outside this contract review.

2.19.4 Proof of concept

Excerpt from LOC03-local-poc.test.ts:217-223:

The local run confirmed both the account closure and reduction in token supply. Unauthorized-operator and active-share controls were rejected.

```
Recorded local reproduction

const supplyBefore = BigInt((await connection.getTokenSupply(baseMint)).value.amount);
const baseRent = (await connection.getAccountInfo(vaultBase, "confirmed"))!.lamports;
const managerBefore = await connection.getBalance(manager.publicKey, "confirmed");
const burnSig = await reclaimTokens(operator, vaultBase, baseMint);
expect(await connection.getAccountInfo(vaultBase, "confirmed")).to.equal(null);
expect(supplyBefore - BigInt((await connection.getTokenSupply(baseMint)).value.amount))
  .to.equal(BASE_DONATION);
```

Selected archived excerpt; the original standalone harness is not included in the package.

2.20 State cleanup can strand a funded unlisted token account

INFORMATIONAL ACKNOWLEDGED

Original assessment: Recorded under the accepted post-exit residual policy. Backend account discovery and final checks are outside this audit.

reclaim_vault_state closes the vault state and asset registry after checking the vault's base-mint account and accounts for mints currently in the registry. It does not require accounts for mints that were never registered or were removed earlier. A funded token account for one of those mints can therefore remain owned by the vault address after the state used by the existing cleanup instruction has been closed.

A source comment assigns discovery of those unlisted accounts to the backend. That backend was not reviewed.

2.20.1 Code location

programs/fbyt_vault/src/instructions/rent_reclaim/reclaim_vault_state.rs:23-42, 77-89

```
#[account(
    mut,
    close = money_manager,
    has_one = admin_pool,
    has_one = money_manager,
    has_one = asset_registry,
)]
pub vault_pool: Account<'info, VaultPool>,

#[account(
    mut,
    close = money_manager,
    has_one = vault_pool,
    seeds = [
        ASSET_REGISTRY_SEED.as_bytes(),
        vault_pool.key().as_ref(),
    ],
    bump = asset_registry.bump,
)]
pub asset_registry: Account<'info, AssetRegistry>,
// ...
let ra = ctx.remaining_accounts;
let token_mint = vault_pool.token_mint;
let mints: Vec<Pubkey> = std::iter::once(token_mint)
    .chain(
        ctx.accounts
            .asset_registry
            .asset_mints
            .iter()
            .copied()
            .filter(|m| *m != token_mint),
    )
    .collect();
require!(ra.len() == mints.len() * 2, ErrorCode::InvalidAccountLength);
```

2.20.2 Impact

The local test closed the vault state while an unlisted vault-owned token account still held 20,000,000,000 base units. The current reclaim_vault_token_accounts instruction then rejected an attempt to recover that account because the vault state no longer existed. The tested credit was made after investors had exited, so active-investor loss was not demonstrated. Recovery through an upgrade or a new mechanism was not ruled out.

2.20.3 Remediation

Acknowledged: The client accepts post-exit residual cleanup under its trusted-operator process, including the need to discover unlisted token accounts.

2.20.4 Proof of concept

Excerpt from LOC03-local-poc.test.ts:229-239:

The test also confirmed that state cleanup was rejected while the registered base token account was still open. The bypass applies to the funded unlisted account.

```
Recorded local reproduction

// The account for a mint outside the registry is omitted from state reclaim.
const stateSig = await reclaimState();
expect(await connection.getAccountInfo(vault, "confirmed")).to.equal(null);
expect(await connection.getAccountInfo(registryAddress, "confirmed")).to.equal(null);
expect((await getAccount(connection, vaultUnlisted, "confirmed")).owner.equals(vault))
  .to.equal(true);
expect(await amount(vaultUnlisted)).to.equal(UNLISTED_DONATION);
await rejected(
  () => reclaimTokens(operator, vaultUnlisted, unlistedMint),
  "AccountNotInitialized"
);
```

Selected archived excerpt; the original standalone harness is not included in the package.

2.21 Positive price exponents are misvalued and large powers can panic

INFORMATIONAL FIXED

Original assessment: The arithmetic behavior was reproduced with synthetic inputs. No affected approved production feed or mint was identified in the recorded surveys.

calc_usd_amount divides by 10 to the absolute value of the oracle exponent. A positive exponent should multiply the price, so this sign loss can severely understate vault value and distort swap and share checks if such a feed is accepted. The same function calculates powers of ten in u64 without a checked range. A 20-decimal mint or exponent magnitude 20 overflows in the tested release profile.

The 28 October feed survey found no positive exponent among 3,913 public Pyth feeds, and the previously observed approved mints had five to nine decimals. The tests establish the arithmetic defect. No approved asset with those profiles was identified in the earlier survey.

2.21.1 Code location

programs/fbyt_vault/src/utils/util.rs:490-500

```
pub fn calc_usd_amount(amount: u64, decimals: u8, price: Price) -> Result<u64> {
    require!(price.price > 0, ErrorCode::InvalidPrice);
    let price_fraction = Fraction::from_num(price.price)
        / Fraction::from_num(10_u64.pow(price.exponent.abs() as u32));
    let amount_fraction =
        Fraction::from_num(amount) / Fraction::from_num(10_u64.pow(decimals as u32));
    let amount_usd =
        (amount_fraction * price_fraction * Fraction::from_num(10_u64.pow(SHARE_DECIMALS as u32)))
            .to_floor();
    Ok(amount_usd)
}
```

2.21.2 Impact

With synthetic exponent +2, the release-profile unit test valued one whole six-decimal token at 19,999 micro-USD instead of the mathematically correct 200,000,000 micro-USD. A devnet-feature local swap using a synthetic price update and a dummy route accepted an adverse exchange; that is not proof that a production Jupiter route or approved feed permits it. The 20-decimal and exponent -20 probes panicked in local test/release arithmetic. Existing-vault liveness for such an asset was not demonstrated.

2.21.3 Remediation

Solved: Positive price exponents are rejected, and unsupported decimal powers return checked errors instead of mispricing or panicking.

2.21.4 Proof of concept

The pinned-source Rust bridge passed in test and release profiles. The selected excerpt checks exponent +1 and a 20-decimal mint; the separate +2 example is retained in the test register. Earlier devnet-feature SBF tests and formal rules FV-31 and FV-32 cover the adjacent paths. No currently approved feed was found to use these exponent or decimal profiles.

```
revision-19a732a/programs/fbyt_vault/tests/security_formal_bridge.rs:46-62

#[test]
fn positive_price_exponent_is_divided_instead_of_multiplied() {
    let price = Price { price: 2, conf: 0, exponent: 1, publish_time: 0 };
    let source_micro_usd = calc_usd_amount(1, 0, price).unwrap();
    // U68F60 rounds 2/10 slightly below 0.2 before the final micro-USD floor.
    assert_eq!(source_micro_usd, 199_999);
    let negative_exponent = Price { price: 2, conf: 0, exponent: -1, publish_time: 0 };
    assert_eq!(calc_usd_amount(1, 0, negative_exponent).unwrap(), source_micro_usd);
    assert_eq!(2u64 * 10 * 1_000_000, 20_000_000);
}

#[test]
fn twenty_decimal_mint_overflows_u64_scale_on_pricing() {
    let price = Price { price: 100_000_000, conf: 0, exponent: -8, publish_time: 0 };
    let observed = std::panic::catch_unwind(|| calc_usd_amount(1, 20, price));
    assert!(observed.is_err(), "expected 10u64.pow(20) to overflow in the test profile");
}
```

IRONNODE

ironnode.io

FBYT / FINAL REPORT

Thank you for choosing IronNode.

PREPARED FOR TELLUS FOUNDATION SA / 24 NOVEMBER 2026