

TroveFi

A non-custodial asset management protocol. Anyone can run a fund or build an index; anyone can invest in either; *nobody* can take the money or stop you leaving.

Networks MegaETH testnet 6343 · Base Sepolia 84532 **Base token** tUSD (6 decimals)

Contracts 15 Solidity sources, non-upgradeable clones **Audit** none

STATUS Live on testnet · unaudited · tokens have no value 329 / 329 Foundry tests passing
every subsystem verified on-chain, on both networks 3 open defects documented in §11

CONTENTS

– Abstract

1 The custody problem

2 System architecture

3 Share accounting

4 Valuation

5 Vaults

6 Buckets

7 Community buckets

8 Manager qualification

9 Fee economics

10 Security model

11 Known limitations

12 Participation record

13 Roadmap

14 Deployment reference

ABSTRACT

Discretionary asset management is gated by custody, not by skill. The licences, prime brokers, auditors and legal wrappers that stand between a competent trader and other people's money all exist to answer one question: who holds the assets, and what makes them give them back. TroveFi removes the question rather than insuring around it. Deposits sit in a smart contract that a manager may trade but cannot withdraw from; the manager's only economic claim is a share of profit above the highest level the fund has ever reached; and every depositor holds an unconditional exit that requires no oracle, no counterparty, and nobody's permission.

The protocol offers three products over one accounting model. **Vaults** are actively managed by a person who has passed an on-chain trading challenge that cannot be talked around. **Buckets** are passive indices with fixed weights and automatic rebalancing, and charge no fee at all. **Community buckets**, new in this version, are buckets anyone can build, which open to other depositors only when their own on-chain record clears a bar written into the contract. All three value positions from time-weighted averages read out of Uniswap V3 pools' own observation buffers, never from spot; all three mint shares through a virtual-offset formula that makes donation attacks and dust residue harmless; and all three are readable in full from the chain.

This document specifies the mechanism, states the fee model in bytecode-level detail, presents measured results from verification transactions on two live deployments, and enumerates the defects that remain open. The last of those is not an appendix. A protocol that publishes its own incidents is the only kind worth evaluating, and §3.2 describes a share-accounting bug that broke a live product in front of real users.

WHAT VERSION 2 ADDS

Community buckets (§7): anyone can deploy a bucket, and it opens to other depositors when a permissionless check finds that its donation-proof, windfall-adjusted track record clears the platform bar and its creator holds a locked stake. **Autopilot** (§7.7): DCA, stop-loss, take-profit and trailing-stop orders that anyone can execute for a capped tip. **Backtesting** (§7.8) against real price history under the contract's own rules. **A second network**: Base Sepolia runs the same contracts as MegaETH testnet (§14). **One manager credential on every chain** (§8.5), at one address with attested mirrors.

The review record moves too: a 329-test suite, an internal adversarial review, and one new open defect that review found (§11.6). The version 2 contracts postdate that review and have not had one (§10.4).

The custody problem

Running money for other people has always been gated. The gate is not there because trading is hard. It is there because custody is hard.

Someone has to hold the assets. Every institution layered around a fund — the licence, the prime broker, the administrator, the auditor, the legal wrapper, the minimum AUM large enough to pay for all of it — exists to make the holder give the money back. That machinery costs several hundred thousand dollars a year to operate before a single trade is placed, which sets a floor on fund size and, by extension, on who is allowed to try.

The result is a filter that selects for capital and connections rather than for skill. A trader with a genuine edge and no network cannot raise; an incompetent one with the right introductions can. Nothing in that arrangement is a law of nature. It is the price of an unsolved custody problem.

1.1 What a contract changes

A smart contract can hold assets and expose a permission surface narrower than any legal agreement can describe. The vault in this protocol grants its manager exactly one capability: swap between the base token and a whitelisted asset. There is no function that sends assets to the manager's address. There is no proxy admin, no upgrade path, and no governance action that could add one, because deployed vaults are minimal-proxy clones of a fixed implementation.

That single constraint dissolves most of the apparatus. If the manager cannot take the money, no auditor is needed to confirm they have not. If the depositor can exit unilaterally, no redemption terms need negotiating. If positions and NAV are readable at every block, no quarterly letter is required. What remains is the only thing that was ever the point:

The manager has exactly one job — be right about the market.

1.2 What a contract does not change

Non-custodial means the money cannot be *taken*. It does not mean the money cannot be *lost*. Nothing in this protocol prevents a manager from putting an entire vault into one asset at the top, churning until fees eat the return, or simply being wrong. The two mitigations are structural rather than financial: every trade is public the moment it happens, and every depositor can leave without asking.

There is no insurance fund, no coverage pool, and no compensation scheme anywhere in this design. Readers looking for one should stop here.

STANDING DISCLAIMER

TroveFi is deployed on MegaETH testnet and Base Sepolia. The contracts have not been reviewed by a third party. Tokens on testnet have no value and are given away by a faucet. Every measured figure in this document comes from those deployments. Nothing here is an offer, a solicitation, or a prediction of returns.

System architecture

Five contracts and one absent edge. The absent edge is the product.

A depositor sends base token to a vault and receives ERC-20 shares. A manager, holding a soulbound credential, may instruct that vault to swap between the base token and any asset the platform registry has whitelisted. Prices come from the pools themselves, averaged over thirty minutes. Nothing else connects.

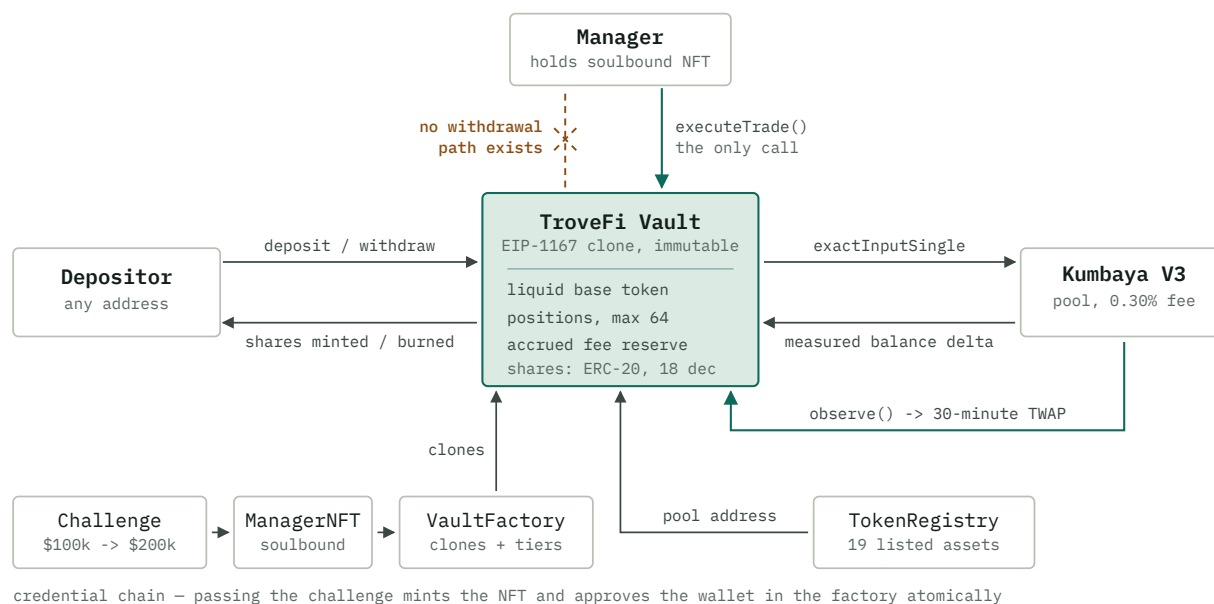


FIGURE 1 The manager reaches the vault through exactly one function. The dashed, crossed edge is the claim the whole design rests on: no function in the deployed bytecode moves principal to the manager, and clones cannot be upgraded to add one.

2.1 The components

The vault holds tokens and issues shares. Share price is holdings divided by shares outstanding. The manager's key opens one door.

The token registry is the platform whitelist. It maps each tradeable token to the Uniswap V3 pool that prices it. Delisting stops new buying immediately but deliberately keeps the pool reference alive, so a vault still holding a delisted token can still value it and still sell it.

The oracle turns pool state into a price. It reads the pool's own observation buffer and returns the time-weighted average tick over a window, thirty minutes by default. It never reads spot.

The factory deploys vaults. It verifies the caller holds a manager credential, reads the base token's decimals (reverting rather than assuming 18 and producing wrong NAV forever), clones the implementation as an EIP-1167 minimal proxy, and initialises it. A clone costs tens of thousands of gas rather than millions.

The challenge and the credential. A prospective manager doubles a virtual \$100,000 against live pool prices. Passing mints a soulbound ERC-721 and approves the wallet in the factory at tier 1, in one transaction. \$8 specifies it.

The **bucket factory**, new in version 2, deploys community buckets for anyone and decides, by fixed criteria, when each opens to other depositors. The **autopilot** executes standing orders on those buckets for whoever calls it. §7 specifies both.

Two networks. The same contracts run on MegaETH testnet and Base Sepolia, each with its own base token, registry, pools and buckets. Nothing is bridged. The one contract shared by address is the manager credential, deployed with CREATE2 so that it sits at the same address on both (§8.5).

2.2 Three products, one accounting model

VAULTS, CURATED BUCKETS AND COMMUNITY BUCKETS COMPARED

	VAULT	CURATED BUCKET	COMMUNITY BUCKET
Who decides holdings	a manager, continuously	fixed weights set by the platform	fixed weights set by its creator
Holdings	any whitelisted token, up to 64	a fixed list, currently 4	2 to 10 registry tokens
Rebalancing	manual, at discretion	automatic on drift, at most daily	automatic on drift, at most daily
Performance fee	1.0%–6.0% of profit above HWM	none	none
Cost to enter	gas only	~0.34%–0.36% in pool fees	~0.30% per leg in pool fees
Withdrawal can fail for liquidity	yes, if fully deployed	no, always liquidates	no, always liquidates
Guaranteed in-kind exit	on close or 7-day pause	no	yes, always
Who can deposit	anyone	anyone	its creator, until it graduates
What you are betting on	one person's judgement	a sector	a basket someone designed

All three share the accounting model in §3, the oracle in §4, 18-decimal ERC-20 shares, and per-address cost basis that travels proportionally when shares are transferred — without which a receiver's entire holding would read as profit.

Share accounting

The section that begins with a bug, because the bug is why the current formula exists.

3.1 The first-deposit problem

An empty pool must price its first share somehow. The naive rule — one share per unit deposited — carries a well-known attack:

1. The attacker deposits 1 wei and receives 1 share.
2. They transfer a large balance directly to the contract, bypassing `deposit()`.
3. The pool now holds a large balance backing exactly one share.
4. The next depositor's shares, computed as $\text{amount} \times \text{supply} / \text{assets}$, round down to zero for anything smaller than that balance.
5. The victim receives nothing and the attacker owns everything.

The traditional fix is to mint a small number of unredeemable "dead shares" to a burn address on the first deposit, so supply can never be as small as one. TroveFi originally did this, with 1,000 dead shares. It was the wrong mechanism for the right problem.

3.2 What went wrong: the BLUE v1 incident

Dead shares survive a full exit. That turns out to matter enormously.

On 29 August 2026 the sole holder of the BLUE bucket withdrew everything they owned. What followed was not an attack. It was the ordinary lifecycle.

1. redeemed 999,999,999,999,999,999,000 of a supply of 1,000,000,000,000,000,000 the 1,000 wei left behind were the dead shares – a true full exit was impossible
2. `withdraw()` paid $\text{balance} \times \text{shares} / \text{supply}$, rounded down with a ratio of $1 - 1\text{e-}18$ and every balance below $1\text{e}18$, this floored to balance – 1
on-chain at that block: `tBTC=1 tETH=1 tSOL=1 tXRP=1`
3. the bucket was left with 1,000 share-units backing 682 base-units of dust
NAV per share jumped from 0.996 to $6.8\text{e}17$
4. the next deposit inherited it:
 $3,971,741,947 \times 1000 / 682 = 5,823,668,543$ share-units
for a 4,000 tUSD deposit where par was $4,000\text{e}18$
a share price roughly $1\text{e}12$ too high

Nothing on that controller could unwind it. The dead shares sit at a burn address; the share price is emergent from the supply-and-portfolio pair; there was no rescale entry point; and the share token could not be reused, because its outstanding units were denominated in the broken scale. The controller was deactivated and replaced.

DISCLOSURE

BLUE v1 (0x4aF5161a...) was deactivated in the registry rather than destroyed, and deliberately left **unpaused** so its three holders could still exit. The broken scale never mis-stated anyone's share of the pool, only the units it was quoted in; every holder retains a correct pro-rata claim.

3.3 The virtual offset

The fix replaces real dead shares with a virtual offset — the ERC-4626 decimals-offset pattern — applied identically in the vault, the bucket controller, and, since version 2, the community bucket.

```
shareScale = 10 ** (18 - baseDecimals) // 1e12 for a 6-decimal base

// minting
shares = amount * (supply + shareScale) / (assets + 1)

// NAV per share, 1e18 == par
nav = (assets + 1) * shareScale * 1e18 / (supply + shareScale)
```

FOUR CHANGES, APPLIED TOGETHER

	BEFORE	AFTER
Seed mint	<code>amount * 1e12 - DEAD_SHARES</code> , minting 1,000 dead shares	<code>valueAdded * shareScale</code> , no dead shares
Pro-rata mint	<code>valueAdded * supply / portfolio</code>	<code>mulDiv(valueAdded, supply + shareScale, portfolio + 1)</code>
"Is it empty" test	<code>supply == 0</code>	<code>supply < shareScale</code>
Full exit	floored proportion, strands dust	sweeps the entire balance
NAV when empty	0	par

The offset is exactly neutral in normal operation. When `supply == assets * shareScale` the terms cancel, and a deposit worth v mints `v * shareScale` — precisely what the old seed branch produced. It changes behaviour only when the portfolio is dust-sized, which is the state it exists to make harmless.

The empty test changed for a reason. Any successful seed mints at least `shareScale` share-units, so a supply below that floor cannot be a live pool. It is the residue of one that was withdrawn out of existence, and pricing a fresh deposit against that residue is what set the broken share price. Re-seeding at par instead costs the residual holder only their sub-unit of dust.

It also closes the original attack. The donation attack works by making `assets` large while `supply` is 1. The seed branch now deliberately ignores the existing balance: a pool with no shares outstanding but a non-zero balance has been donated to, and pricing the first share against that donation is the attack. The seed always prices at par, whatever has been sent in.

3.4 The full-exit sweep

Ordinary withdrawals round down, leaving a sub-unit behind each time. Left to accumulate, a pool could reach zero shares while still holding stranded units, and those units would re-price the next depositor — exactly the BLUE v1 failure.

So the final withdrawal, the one burning the entire remaining supply, sweeps the whole balance rather than a floored proportion. In a vault this is gated on there being no open positions, because a sole holder's claim includes those positions and paying out only the base would abandon them. In-kind redemption always sweeps, because it moves the held tokens themselves and cannot abandon anything.

MEASURED ON-CHAIN – ALL FIVE BUCKETS EMPTIED AND RE-SEEDED

BUCKET	STRANDED UNITS AFTER FULL EXIT	RE-SEED RATIO
BLUE	0	1.000000
AIML	0	1.000000
DEFI	0	0.999945
RWA	0	1.000000
PAYFI	0	0.999930

The old code left one unit of each of four assets in this exact scenario, and BLUE v1's re-seed was off by a factor of about $1e12$. Two invariants now guard it in CI:

```
invariant_deadSharesNeverMinted and  
invariant_navStaysWithinAnOrderOfMagnitudeOfPar .
```

3.5 The general lesson

The vault and the bucket carried the same defect for the same reason, and the vault's copy survived the bucket's fix because the two were treated as separate codebases rather than as one accounting model implemented twice. Both now use the same formula, and the tests that catch it run against both. The residual cost of that mistake is documented in §11.1: vault clones deployed before the fix are immutable and keep the old bytecode.

Valuation

Every price in the system comes from one place: the pool's own observation buffer, averaged over time. There is no external feed anywhere.

4.1 Why not spot

A V3 pool's `slot0()` returns whatever the last trade left behind. Anyone with enough capital, or a flash loan, can move it arbitrarily far inside a single block and move it back in the same transaction. If NAV read spot, the attack writes itself: flash-loan a token the vault holds, spike the pool, transact against the vault at the distorted valuation, trade back, repay. Every share mint, every fee accrual, and every slippage floor would be exploitable the same way.

4.2 What is read instead

V3 pools maintain a ring buffer of cumulative tick observations, written by swaps as they happen. Reading two observations n seconds apart and differencing them yields the arithmetic-mean tick over that window, which converts to a time-weighted average price.

```
consult(pool, period):
    observe([period, 0])          -> two cumulative ticks
    meanTick = (tick[1] - tick[0]) / period -> rounded toward -infinity
    require MIN_TICK <= meanTick <= MAX_TICK
```

To move that average, an attacker must hold the price away from the market for the whole window while every arbitrageur on the chain trades against them. It is not a single-block trick but a sustained, capital-intensive, publicly visible position that leaks money continuously, and for any realistic fund size it costs far more than it can extract.

TWAP WINDOWS

WHERE	WINDOWWHY
Vault NAV, trades, fee accrual	1,800 s long enough to make manipulation expensive, short enough to track the market
Curated bucket NAV, deposits, withdrawals, rebalance	300 s live testnet setting; the contract default is 1,800 s
Community bucket NAV, swaps, checkpoints, Autopilot triggers	300 s one factory setting read by every community bucket
Manager challenge	300 s trading windows are short; five minutes still defeats a flash loan

Vault, curated bucket and community bucket windows are adjustable by the platform owner between 60 seconds and 24 hours. The challenge window is adjustable within the same range, and its setter **rejects zero**, so the manipulable spot path is unreachable even by misconfiguration. That is deliberate: passing the challenge auto-approves someone to run a fund holding real deposits, so the manipulable path must not exist at all.

4.3 The NAV calculation

```
assets      = baseBalance - accruedManagerFee
              + sum(positionBalance × twapPrice)   for each held token

navPerShare = (assets + 1) × shareScale × 1e18 / (supply + shareScale)
```

The accrued manager fee is subtracted because it is already owed and is not part of what backs depositor shares. An empty vault returns par by definition rather than dividing by zero or reporting a ratio derived from leftover dust.

Two view functions, `previewDeposit()` and `previewWithdraw()`, mirror the real arithmetic exactly — including which branch it takes — and match the transaction to the raw unit. A preview that disagreed with the mint would have the interface quoting a number the chain never produces.

WHAT MOVES NAV PER SHARE, AND WHAT DOES NOT

EVENT	EFFECT ON NAV PER SHARE
Trading results	raises or lowers it — this is the entire point
Price moves in held assets	marked to TWAP continuously, whether or not the manager trades
Swap costs	0.30% pool fee plus price impact, borne by NAV, so borne by depositors
Fee accrual	drops it by the fee; the HWM is set to the <i>pre-fee</i> NAV so the fee creates no phantom high
Deposits	no movement — measured at 0.00000% on a live 2,000 tUSD deposit
Withdrawals	no movement; rounding leaves a sub-unit that nudges NAV a fraction of a bp in remaining holders' favour
Share transfers	no movement — nothing about the pool changes

4.4 Cost and bounds

Every NAV read walks the vault's *own* open positions, not the platform whitelist. A vault running three positions performs three oracle reads regardless of how many tokens the platform has ever listed. Simultaneous open positions are capped at `MAX_HELD_TOKENS = 64`.

MEASURED DEPOSIT GAS BY OPEN POSITION COUNT

HELD TOKENS	DEPOSIT GAS	MARGINAL PER TOKEN
1	73,644	—
8	154,753	11,587
32	450,566	12,158
64	846,240	12,263

About 12,300 gas per position, flat, with no hidden superlinearity. At the cap a deposit costs 11.5× the single-position case, roughly 93% of it inside that loop. Bounded and safe — but the manager chooses the position count and every depositor pays for that choice on every transaction. See §11.4.

4.5 Pool requirements and failure modes

Observation cardinality. A pool must hold enough observation slots to cover the window being read. TroveFi's testnet pools are expanded once, off-chain, to cardinality 20 on MegaETH and 60 on Base Sepolia.

Fresh enough buffer. If the requested window extends past the oldest stored observation, `observe()` reverts with `OLD` and the whole calling transaction reverts. That is the correct failure mode: a deposit, withdrawal, or NAV read must not silently fall back to a stale price.

A real V3 pool. The registry probes V3-only methods at listing, so a V2 pair — which does not implement `observe()` — can never be registered. A V2 pair as a price source would permanently break deposits and withdrawals on any vault that ended up holding that token.

Checking pool health correctly. The manager dashboard checks pools by calling `observe([twapPeriod, 0])`, the exact call the vault makes. An earlier version checked observation cardinality against a threshold and reported a permanent false alarm: cardinality tells you how many slots a pool has, not whether the window you actually read is available, and observations are written only when a pool is touched, so a quiet pool at cardinality 20 can span hours comfortably. The only reading that predicts whether a transaction will succeed is the call that transaction will make.

4.6 What TWAP costs

Lag. In a sharp move, TWAP-priced NAV reads below true value in a rally and above it in a crash, converging once the move settles. Consequences worth stating:

- Depositing during a sharp rally means buying at a NAV that has not caught up, which slightly favours the depositor over existing holders. Withdrawing during one is the reverse.
- A manager's TWAP-floored minimum-out can become the binding constraint during fast moves, causing a trade to revert. That is the floor working.
- Fee accrual lags the same way: a gain is charged after the average catches up, not at the peak tick.

This is a deliberate trade. A number that lags by minutes is worth far more than a number anyone can forge in one block.

MEASURED ON-CHAIN – ORACLE CORRECTNESS, ALL 19 LISTED ASSETS

On-chain NAV price equals the TWAP recomputed off-chain from raw `observe()` ticks using an independent port of the tick math. Spot matches TWAP to 0.000% on every pool, fully converged. All nineteen pools have liquidity, cardinality 20 of 20, and `observe([300, 0])` does not revert. Portfolio value equals the sum of asset values plus idle base, exactly.

Resistance was demonstrated directly: when a pool price was moved from \$149.98 to \$327.54 for testing, the 300-second average was watched converging toward it over time rather than jumping.

Vaults

A pot of money with one trader attached and a contract in between them.

The depositor sends base token and receives shares. The manager trades the pot. Share count does not change; share value moves with results. On exit, shares burn and base token comes back. The manager never holds the money.

5.1 The permission surface

COMPLETE MANAGER PERMISSION SET

CAN	CANNOT
Open a position — swap base into any token active in the registry, while LIVE	Withdraw, transfer, or borrow against depositor funds
Close a position — swap any held token back to base, in <i>every</i> state, including for delisted tokens	Trade a token that is not on the registry whitelist
Set a minimum-out per trade, subject to an enforced floor	Trade token to token without crossing the base token
Sync a holding — bring in a balance that arrived by airdrop or direct transfer	Accept a fill worse than 5% below the TWAP price
Claim accrued fees with <code>claimManagerFee()</code>	Spend the accrued fee reserve
Create more vaults, up to the tier limit	Change the fee schedule — compile-time constants
—	Change their own tier, or hold more than 64 positions
—	Pause, unpause, or close the vault
—	Block or delay a withdrawal, or prevent an in-kind redemption
—	Upgrade the vault — clones have no proxy admin

5.2 The trade path

```
executeTrade(tokenIn, tokenOut, amountIn, minAmountOut)
  require caller is manager
  require exactly one side is the base token
  if buying: require state == LIVE and asset is active in registry
  if selling: require the asset has a known pool // delisted is fine
  if spending base:
    require amountIn <= balance - accruedManagerFee // fee reserve is off limits

  expectedOut = TWAP quote for amountIn
  hardFloor = expectedOut × 95%
  effectiveMin = max(minAmountOut, hardFloor) // whichever is stricter

  balanceBefore = balanceOf(tokenOut)
  swap via router with amountOutMinimum = effectiveMin
  received = balanceOf(tokenOut) - balanceBefore // measured, not reported
  require received >= effectiveMin

  update the held-token book
  emit TradeExecuted(..., received)
```

Three properties of that path are worth isolating.

Every trade crosses base. A manager cannot swap tBTC directly for tETH; they sell to base, then buy. This keeps valuation to one hop per position and makes the whitelist actually constrain what a vault can hold.

The router is not trusted. The vault measures its own balance before and after and credits the difference. The router is the one component in that path the platform can swap out, so its return value is treated as advisory.

The 5% band is a backstop, not a target. It exists to stop a manager submitting a nonsensical minimum and letting a sandwich take the difference. In practice the application quotes each trade through the DEX quoter and submits a 1% tolerance, failing closed if the quote is unavailable rather than falling back to a loose bound. Submitting zero and relying on the contract floor would take the loosest bound allowed on every trade, and that spread comes out of depositors' NAV.

5.3 Lifecycle and states

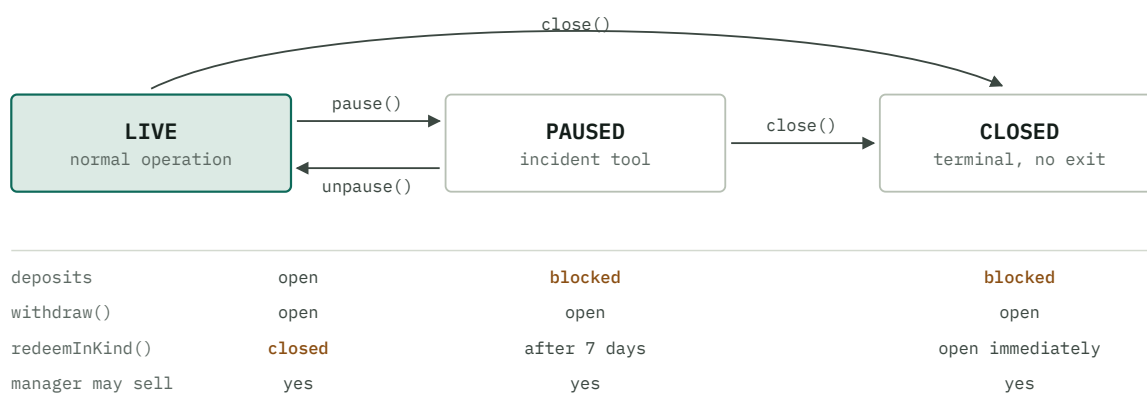


FIGURE 2 Withdrawals never stop. Closing means "no new money and no new exposure", never "you cannot have your money back" — and both exit routes open the moment a vault closes.

Creation. A credentialled manager calls `createVault()` with a base token, name and symbol. The vault starts LIVE at tier 1, at par, with a 1,800-second TWAP window, the platform registry, and the default router. Vault count is capped by tier: 1, 2, 5, unlimited.

Pausing is an owner power, not a manager one. It blocks deposits and new positions while leaving withdrawals working. The seven-day clock is the important part: a pause with no time limit would freeze funds just as effectively as a permanent close, so after seven days holders can take the underlying directly and stop depending on anyone. `timeUntilInKindRedemption()` returns the countdown; unpausing clears it. That reset is also a weakness: an owner who cycles pause and unpause inside every seven days can keep the clock from ever running out (§11.6).

Closing is terminal. There is no forced liquidation and no auto-close: a closed vault holding positions keeps holding them until the manager sells them back to base or holders redeem in kind. Every transition emits `VaultStateChanged` with previous state, new state, and timestamp.

5.4 Exit guarantees

The promise is narrow and precise: nobody can stop you leaving. There are two routes because the one you want in normal operation has a dependency, and an exit guarantee cannot have dependencies.

`withdraw(shares, minBase)` burns shares and pays $\text{shares} \times \text{assets} / \text{supply}$ in base token, in every state including CLOSED. It depends on the vault holding enough liquid base. When it does not, the transaction reverts with `InsufficientLiquidity` rather than paying a partial amount — paying less than the claim while burning the full share count would silently transfer value from the withdrawer to everyone still in the vault.

`redeemInKind(shares)` burns shares and transfers a pro-rata slice of every asset the vault holds, as the actual tokens. It requires no swap, no oracle read, no counterparty, no manager action, no platform action, and no governance decision. It reads token balances, computes the proportion by *share count*, burns, and transfers. Because it is priced by share count rather than NAV, it cannot be gamed by manipulating a TWAP and does not disadvantage whoever redeems last.

It is deliberately closed while a vault is LIVE. Letting anyone pull tokens out of a live vault in kind would fragment its positions, leave the manager holding odd remainders, and hand a redeemer the choice of taking assets out at a moment of their choosing. The guarantee is not "you can always take tokens"; it is "you can always get your money out".

TWO EXCEPTIONS THAT CLOSE A FREEZE

Selling is permitted in every state, whitelisted or not. Gating sales on the whitelist would leave a delisted position permanently unsellable. Gating them on LIVE would mean a paused or closed vault could never be liquidated at all, so holders could not be paid in base even in principle. The whitelist and the LIVE state govern what a manager may *buy*, never what they may sell.

Delisting does not erase value. Valuation goes through `poolOf()` rather than through the active flag, so a vault still holding a delisted token keeps its NAV contribution. Before this fix, delisting a held token silently erased its entire value from NAV in a single admin transaction.

A GAP IN THE EVIDENCE

`redeemInKind()` has been exercised on a live closed vault, correctly returning base pro-rata and burning shares to zero. That vault held only base token, so the leg that returns a *non-empty* list of tokens and amounts is covered in the test suite but has not yet run on-chain against a vault holding multiple positions.

HISTORICAL NOTE

An earlier version of the contract gated `withdraw()` on the vault not being closed. Combined with closing being terminal, that meant a single owner transaction could permanently freeze every depositor's funds with no redemption path at all. It was found in internal review and fixed.

Buckets

An index that runs itself. Fixed holdings, fixed weights, automatic rebalancing, no manager, no performance fee.

Pick a theme, pick four tokens that represent it, assign weights summing to 10,000 basis points, and deploy a contract that holds exactly those tokens at exactly those weights and puts them back when they drift. That is the entire product. There is no discretion in it and nothing to trust beyond the code.

This section describes the five **curated** buckets the platform runs, the same five on both networks. Community buckets, which anyone can build, share the idea and differ in the ways \$7 sets out.

6.1 Deposit and withdrawal accounting

A bucket deposit is one transaction that does real work: it pulls the base token, splits it by weight, swaps into each holding, re-values the portfolio, and mints shares against the value that *actually arrived*.

```
deposit(amount)
  1. snapshot portfolio value and share supply
  2. pull base token, verify the exact amount arrived // fee-on-transfer -> IncorrectTransfer
  3. for each holding: amountIn = amount × weight / totalWeight
      swap with a TWAP-derived minimum-out, 300 bps tolerance
  4. re-value the portfolio at TWAP
  5. require the portfolio grew by >= 96% of the deposit // else DepositSlippage
  6. shares = valueAdded × (supply + shareScale) / (portfolioBefore + 1)
  7. cost basis += the full amount sent; blend weighted entry timestamp by cost
```

Why credit value added rather than face value. Swap costs are real and they land on the incoming deposit, not on existing holders. Minting against `valueAdded` puts that cost where it belongs; crediting face value would take the difference out of everyone else's share price.

Why PnL starts slightly negative. Shares are minted against value added, but cost basis records what was actually sent. The new holder is down the entry cost — about 0.35% — until the holdings move enough to cover it. That asymmetry is honest accounting, not a glitch.

Withdrawal reverses the sequence: snapshot, compute the TWAP-implied claim as `portfolio × shares / supply`, adjust cost basis proportionally, burn, liquidate the proportion of each holding with per-swap floors, require the total returned to be at least 96% of the implied claim, and pay.

There is no liquidity condition. A bucket always liquidates on the spot, so it has no equivalent of a vault's `InsufficientLiquidity`. The realistic failure mode is the opposite: a withdrawal large enough to move the pools past the 4% guard reverts on `WithdrawSlippage`. The holder is protected from a bad fill rather than blocked from leaving, and splitting the withdrawal gets them out.

6.2 Rebalancing

Left alone, an index stops being the index: the winner grows into an outsized share and the thing you bought quietly becomes a concentrated bet on whatever went up most.

Two conditions, both required. At least `rebalanceInterval` since the last one — one day by default, never less than an hour — and the worst-drifted holding must exceed `driftThresholdBps`, 200 bps by default. If either fails, the call reverts harmlessly with `RebalanceTooSoon` or `DriftBelowThreshold`.

Drift is measured relative to target, not in absolute points. A holding targeted at 25% now sitting at 27% has drifted $2 / 25 = 800$ bps, not 200. A five-point move in a 30% holding and a five-point move in a 10% holding are very different distortions, and relative measurement treats them as such.

The trigger is permissionless. `rebalanceIfNeeded()` can be called by anyone: a holder, a bot, a bystander. The conditions are checked on-chain, so nobody can force one early, and a bucket does not depend on the team being awake.

Execution values the portfolio at TWAP, computes each holding's target value, sells every overweight holding back to base, then buys every underweight holding with the base now available. Selling before buying means the base needed for purchases is already there — no flash loan, no intermediate borrowing, no external dependency.

THE GUARDS, ALL ARMED AT CONSTRUCTION

GUARD	CONTRACT DEFAULT	LIVEWHAT IT CHECKS
Per-trade minimum out	100 bps	300 bpsevery swap carries an <code>amountOutMinimum</code> from the TWAP quote
Deposit	100 bps	400 bpportfolio value must rise by at least 96% of what was deposited
Withdrawal	100 bps	400 bpspayout must be at least 96% of the TWAP-implied claim
Rebalance	100 bps	400 bpportfolio may not lose more than 4% across the whole rebalance
Value creation	1e-4 base	1e-4 baseportfolio may not end up worth measurably <i>more</i> than it started

`guardsArmed()` returns whether all three global guards are set, so the state is monitorable. Every bps parameter is capped by `MAX_BPS_CAP = 1,000`. The live guards were widened from 100 bps on 7 September 2026: the 0.30% pool fee alone used a third of a 1% budget before any price movement, and ordinary volatility made deposits fail.

The value-creation bound is the unusual one. Rebalancing moves value between holdings; it never creates any. A rebalance that appears to have made the portfolio richer has not found free money — it has read a price wrong somewhere. Reverting on apparent profit sounds paranoid until you consider that the alternative is silently trusting a broken oracle.

Two further correctness rules: **no duplicate assets**, because configuring the same token twice would double-count its balance in NAV and inflate the share price, so it is rejected at configuration time; and **pool cache invalidation**, because each asset's pool address is memoised and a reconfigured fee tier must clear it, or valuation and execution could end up pointing at different pools.

WHAT REBALANCING DOES NOT DO

It will not sell into a crash — if a whole theme falls together, weights stay roughly correct and no rebalance fires. It will not take profit; trimming a winner back to target buys the other holdings rather than going to cash. It will not fire on a small move, and it will not run twice in a day, because rebalancing on every wiggle would burn more in pool fees than the correction is worth.

6.3 The five live buckets

Every constituent was selected under one stated constraint: prefer assets that actually exist on Base, so a bucket could be deployed on a real network without a bridge. Where an asset fails that test it is recorded explicitly rather than quietly included.

LIVE BUCKETS AND CONSTITUENTS

TICKER	THEME	CONSTITUENTS AND WEIGHTS	BASE-AVAILABILITY GAP
BLUE	Crypto Blue Chips	BTC 25 · ETH 25 · SOL 25 · XRP 25	none (via cbBTC, native, uSOL, cbXRP)
AIML	AI & Compute	TAO 30 · VIRTUAL 25 · NEAR 25 · ICP 20	none; RENDER, FET, GRT dropped for having no Base contract
DEFI	DeFi Blue Chips	AAVE 30 · MORPHO 25 · ENA 25 · PENDLE 20	none — all four native on Base; UNI omitted for lack of one
RWA	Real World Assets	LINK 35 · ONDO 30 · PAXG 20 · SYRUP 15	ONDO and PAXG have no Base contract
PAYFI	PayFi & Settlement	XRP 30 · XLM 25 · LTC 25 · HUMA 20	XLM and HUMA have no Base contract

RWA is the only bucket that could not be built entirely on Base, and the reason is worth stating: Base has the tokenised-fund side (BUIDL, Spiko's EUTBL and USTBL, Janus's JTRSY and JAAA), but those instruments have effectively no secondary volume and no exchange feed, while the assets that actually trade have no Base contract. A sixth bucket — an L2 index of ARB, OP, POL and STRK — was drafted and shelved because all four fail the same test.

TWAP WINDOW 300 s contract default 1,800 s	REBALANCE INTERVAL 1 day minimum 1 hour	DRIFT THRESHOLD 200 bps relative to target	PER-TRADE SLIPPAGE 300 bps contract default 100 bps
POOL FEE TIER 3000 0.30% per leg	PERFORMANCE FEE none		

MEASURED ON-CHAIN — MECHANISM PERFORMANCE

BUCKET	ENTRY COST	ROUND TRIPNAV MOVED BY A 500 TUSD	D
BLUE	0.360%	0.590%	0.00000%
AIML	0.340%	0.590%	0.00000%
DEFI	0.340%	0.590%	0.00555%
RWA	0.360%	0.590%	0.00000%
PAYFI	0.340%	0.590%	0.00000%

The Kumbaya pool fee is 0.30% per leg, so a two-leg round trip has a floor of 0.60% before rounding; buckets come in fractionally under it. In every case the pre-existing holder's claim went slightly *up* after another user's round trip — the withdrawer's floored rounding stays in the pool, which is the correct direction. No deposit diluted anyone. A separate test rebalance corrected 704 bps of drift back to 4 bps, restoring weights to [2499, 2500, 2500, 2500], at a cost of 0.014% of NAV.

None of that cost is a TroveFi charge. It is the DEX, and assembling the same basket by hand would cost the same 0.30% per swap plus four times the gas, with manual rebalancing thereafter.

Community buckets

New in version 2. Anyone can build a bucket. Nobody can list one: it lists itself, when its own record clears a bar written into the factory.

A curated bucket is open from its first block because the platform stands behind the selection. That model does not extend to user-built baskets without an approval process, and an approval process is discretion, which this protocol removes everywhere else. So version 2 separates building from listing. `BucketFactory` deploys a bucket for anyone. The bucket starts **private**, accepting deposits only from its creator. It becomes **public** when a permissionless call to `graduate()` finds that its track record, measured on-chain, clears the platform's bar. There is no review queue and no allowlist. The criteria are the approval.

Everything therefore rests on the track record being expensive to fake, and most of this section is about why it is.

7.1 The contracts

CustomBucket is the bucket: an ERC-20 whose shares claim a fixed-weight basket of registry tokens. Each is an EIP-1167 clone of one implementation. It has no fee field and no function that moves assets anywhere except to a holder burning shares or to the DEX in a swap. At 23,988 bytes of runtime code it sits just under the EIP-170 limit, which is why its portfolio and preview views live on the factory.

BucketFactory deploys buckets, holds the policy every bucket reads on each call, decides graduation, and serves the views the bucket has no room for. Its owner can tune policy within hard bounds, pause new deposits across every bucket, and delist a bucket from public view. It cannot move funds, change a composition, or block an exit.

BucketAutopilot holds standing orders on buckets and executes them for anyone who calls, paying the caller a capped tip (\$7.7).

BUCKET POLICY, IDENTICAL ON BOTH NETWORKS

RULE	TESTNET VALUEHARD BOUND IN THE FACTORY
Constituents	2 to 101 to 16
Minimum weight	5%at least 1 bp, and minimum × count ≤ 100%
TWAP window	300 s60 s to 24 h
Per-swap slippage allowance	300 bps1 to 1,000 bps
Rebalance drift threshold	500 bps, relative1 to 10,000 bps
Rebalance interval	1 dayat least 1 hour
Reweight notice	1 day0 to 30 days
Checkpoint interval	6 hoursat least 10 minutes
Maximum lock	365 days1 day to 4 years
Minimum deposit	1 tUSDunbounded

Policy is read live from the factory, so a change reaches every existing bucket at once. That lets a bad parameter be corrected without migrating anyone. It also lets the owner change the terms of buckets people already hold; §11.8 lists what that allows.

7.2 Accounting that cannot be donated to

A curated bucket values its tokens by reading its balances. A community bucket cannot, because anything that reads `balanceOf()` can be raised by sending tokens to it, and a record that can be raised by a transfer is not a record. So a community bucket keeps its own book. `heldBase` and `heldBalance[token]` change only when its own deposits, swaps and exits move them, and NAV is priced off that book at TWAP. Tokens sent from outside never enter it. The owner can sweep such strays with `skim()`, which by construction can only take the excess over the accounted balance.

```
deposit(amount, minShares, unlockAt, receiver = caller)
1. private bucket: receiver must be the creator          // else NotPublic
2. value the book at TWAP; record a checkpoint if due
3. pull base, require the exact amount arrived          // else IncorrectTransfer
4. for each weighted, still-listed constituent:
    swap amount × weight, minOut = TWAP × (1 - fee) × (1 - 3%)
    credit the measured balance delta to the book
5. valueAdded = value after - value before              // else NoValueAdded
6. shares = empty ? valueAdded × shareScale
    : valueAdded × (supply + shareScale) / (valueBefore + 1)
7. cost basis += amount; optionally lock exactly these shares
```

That is the accounting model of §3 with the private book added. A slice for a constituent the registry has since delisted is not swapped and stays in the book as base. Withdrawal is the reverse: the leaver's slice of each holding is sold at a TWAP-floored price, and the leaver bears their own swap costs, so remaining holders are never diluted by an exit. The last holder out takes the whole book.

7.3 The track record

A bucket writes a **checkpoint**, a timestamp and a NAV per share, at most once per checkpoint interval. Any deposit, withdrawal or rebalance writes one when it is due, and anyone can call `checkpoint()`. A record belongs to a *life*. The first deposit into an empty bucket seeds it at par and starts one; the last holder leaving ends it and strips public status. Earlier lives stay readable on-chain but never count.

The number checkpointed is not raw NAV. It is:

```
trackNav = navPerShare × windfallFactor / 1e18

windfallFactor = 1e18 at the start of each life, and on every rebalance that gains value:
windfallFactor = windfallFactor × valueBefore / valueAfter    // never on a loss
```

The windfall factor closes the one manipulation a donation-proof book leaves open. A rebalance values the bucket at TWAP but trades at spot. A creator who pushes a pool's spot price away from its average and then triggers a rebalance into it can make the swaps fill better than TWAP, so the bucket ends the rebalance worth more than it began. That gain is real, and holders keep it. The record does not: the factor divides it back out, and `WindfallExcluded` records each time it does. Losses are never excluded, so a rebalance can only count *against* the record. The asymmetry is deliberate. Excluding losses too would let a creator bury a bad rebalance; excluding neither would let them buy a good one.

7.4 Graduation

WHAT `GRADUATE()` REQUIRES, ALL AT ONCE

CRITERION

TESTNET VALUE

WHAT IT PREVENTS

CRITERION	TESTNET VALUE	WHAT IT PREVENTS
Age of the current life	at least 3 days	an overnight record
Each of the last k checkpoints	$k = 4$, each $\geq \text{par} \times 1.05$	one good day, real or manufactured
Oldest of those k	no more than 30 hours old	a good streak from long ago
Live track NAV	$\geq \text{par} \times 1.05$ at the moment of graduation	graduating on the way down
Creator's locked stake	worth at least 500 tUSD	a creator with nothing at risk
Creator's lock remaining	at least 7 days	listing and leaving in the same week
Not banned	not delisted by the owner	an abusive bucket relisting itself

Testnet values, short enough to watch a full cycle in under a week. The deploy script records a mainnet starting point of a 30-day record, seven daily checkpoints inside eight days, +10%, and a creator lock with 90 days left. `graduationStatus(bucket)` returns every criterion separately, so the interface can show a creator exactly what is missing.

`graduate()` is permissionless, and records any checkpoint that is due before it judges, so a bucket is measured on its latest state. There is no deadline: a bucket that never qualifies simply stays private. The owner's one lever over an individual bucket is `delist()`, which returns it to private and bans it from graduating until the owner lifts the ban. Delisting exists for abuse, such as a misleading name, and touches nothing else. Every holder keeps both exits.

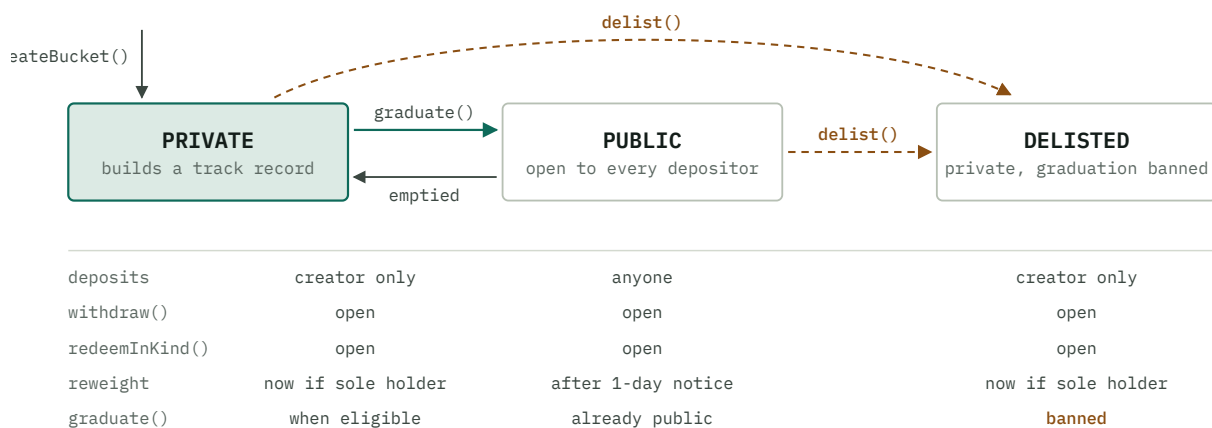


FIGURE 3 A community bucket lists itself. `graduate()` needs no approval, only the criteria in the table above; `delist()` is the one owner action over an individual bucket, and it never touches an exit. A bucket that empties loses public status and starts a new record from par on its next seed. A reweight applies at once only while the creator holds every share of a private bucket; otherwise it waits out the notice.

7.5 Composition changes and locks

Rewighting. The creator can propose a new basket at any time. It is validated against the same rules as creation when it is proposed, and again when it is applied. While the bucket is private and the creator holds every share, it applies at once, because nobody else is affected. Otherwise it is queued for the reweight notice, one day on testnet, readable on-chain through `pendingReweight()`, and applicable by anyone once the notice has run. Every other holder sees a composition change coming and can leave before it lands. A dropped token is not dumped in the same transaction. It stays in the book at zero weight, still valued, and counts as full drift, so the next rebalance sells it; a new basket also resets the rebalance clock, so that rebalance is available at once.

Locks. Any holder may lock their own shares until a date, at most a year out. A lock can be extended and never shortened. It is enforced in the token's transfer hook, so locked shares cannot be withdrawn, redeemed, transferred or escrowed. And only a holder can lock their own shares: a deposit that locks must mint to its caller, because otherwise a stranger's one-unit deposit could extend someone else's lock. Locks exist for the creator-stake criterion. They are also the one thing that can stop a holder leaving, which is why the owner has a single emergency power over them, `releaseLocks()`, which frees every holder of one bucket at once and permanently, and cannot create or extend anything.

7.6 Exits

Community buckets were written after the vault's exit guarantees and take them further. `withdraw()` works in every state, and the owner has no power to pause it; the owner can pause only *new deposits*, and only across every bucket at once. `redeemInKind()` is open **always**: no state to wait for, no clock, no oracle, no router, no factory call. It burns shares and transfers the holder's exact slice of the book. The only thing that can block either route is the holder's own lock.

EXIT ROUTES COMPARED

	VAULT	COMMUNITY BUCKET	CURATED BUCKET
Priced withdrawal	every state	every state	stops while paused
In-kind redemption	on close, or 7 days paused	always	not available
Owner can pause exits	no	no	yes
What can block a holder	liquidity, until in-kind opens	only their own lock	a pause

7.7 Autopilot

`BucketAutopilot` holds four kinds of standing order on community buckets and executes any of them for whoever calls `execute(id)` while its condition holds.

ORDER TYPES

ORDER	FIRES WHEN	WHAT IT DOES
DCA	each interval, at least an hour apart, for n buys	pulls one buy from the owner's wallet and deposits it for them
Stop-loss	$\text{NAV per share} \leq \text{trigger}$	withdraws the escrowed shares and pays tUSD
Take-profit	$\text{NAV per share} \geq \text{trigger}$	withdraws the escrowed shares and pays tUSD
Trailing stop	$\text{NAV} \leq \text{peak} \times (1 - \text{trail})$, trail 0.01% to 50%	withdraws the escrowed shares and pays tUSD

Triggers read TWAP-priced NAV, the bucket's own `navPerShare()`, so pushing spot for a block cannot fire a stop. **Every fill is floored** at the factory's TWAP preview less the owner's slippage tolerance, on top of the bucket's own per-swap floors. **DCA is not prepaid**: funds stay in the owner's wallet under an allowance, and a buy that finds the wallet short simply does not run. **Exits escrow shares** in the contract, and the owner can reclaim them with `cancel()` at any time; cancel is never paused. **Only the owner can cancel**, and the contract owner's only power is the tip.

The tip is paid to the executor out of each execution: 10 bps on testnet, capped in bytecode at 50 bps. It is what makes execution permissionless in practice. TroveFi runs a keeper, and if it stops, anyone who wants the tip can take its place; `isReady(id)` and `readyOrders(offset, limit)` exist so a keeper can scan

without reverting. The cost of that design is that timing is not guaranteed. An order fires when someone executes it, and a stop fires minutes behind a sharp fall because TWAP does (\$11.9).

7.8 Backtesting before building

The builder replays a proposed basket over up to a year of daily closes for the real assets the testnet tokens track, fetched from Binance with Binance US and Coinbase as fallbacks, the same venues the testnet peg bot holds pools to. The simulation runs in the browser and applies the contract's own rules: the pool fee on every leg, a rebalance only when relative drift reaches the live threshold and the live interval has passed, sells before buys. It draws the basket against the same basket never rebalanced and against BTC, with total return, maximum drawdown, annualised volatility, rebalance count and fees paid.

It cannot model price impact beyond the fee, anything inside a day, or TWAP lag. And a basket chosen because it backtested well has been selected on the window it is being judged on. It is a check on a composition, not a forecast.

MEASURED ON-CHAIN: VERSION 2 AT LAUNCH

Deployed to both networks on 16 September 2026 and exercised end to end on each: buckets created and seeded, deposits and withdrawals, and a keeper run against live orders in which a DCA buy and a stop-loss both filled with every balance reconciling exactly. The deployed runtime bytecode matches the repository source on both chains. As of 19 September 2026, seven community buckets and sixteen Autopilot orders exist across the two networks, all from test wallets, and none has yet graduated.

Manager qualification

Anyone can claim a track record. Screenshots are free and a selectively presented wallet history proves nothing. The gate is a trading test that cannot be talked around.

8.1 The simulator

A prospective manager registers once per wallet and receives \$100,000 of virtual balance, held on-chain in whole cents as a `uint64`. They trade it against the platform's listed assets. Nothing they trade is real — balances live in the challenge contract, no tokens move, and no pool is touched by their activity. What is real is the price.

Every price is the time-weighted average tick over the last 300 seconds of a live Uniswap V3 pool, converted into cents per whole token. No Chainlink, no Pyth, no signed message from a server, no relayer. Token balances are held at 1e18 internal precision, independent of the real token's decimals. Every trade must cross base, and every trade pays 30 basis points on the way in and on the way out.

8.2 The gates

CHALLENGE PARAMETERS

PARAMETER	DEFAULTCONFIGURABLE
Starting balance	\$100,000no — compile-time constant
Target balance	\$200,000no — compile-time constant
Minimum duration	7 daysyes, by owner
Minimum trades	10yes, capped at 1,000
Minimum trade size	\$100yes
Minimum hold time	60 syes, capped at 24 hours
Trade fee	30 bpsyes, capped at 500 bps
Minimum pool liquidity	1,000yes
TWAP window	300 syes, between 60 s and 86,400 s — never zero

The target, the trade count and the duration must all hold *simultaneously* at the moment of claiming.

`claimEligibility()` re-checks every gate at call time, then marks the participant claimed, records their final value, mints the ManagerNFT, and calls into the vault factory to approve them at tier 1.

8.3 What each rule blocks

REJECTION PATHS — EVERY ROW EXERCISED AGAINST THE LIVE DEPLOYMENT

ATTEMPT	CONTRACT ERROR
Claim with 0 trades and 0 elapsed time	DurationNotMet
Claim after the duration, 12 trades, below target	TargetNotReached
Sell within 60 seconds of buying	HoldTooSoon

ATTEMPT	CONTRACT ERROR
Two trades in one block	OneTradePerBlock
Trade below the \$100 minimum	BelowMinTrade
Token to token, bypassing base	MustCrossBase
Spend more base than held	InsufficientBalance
Trade an unlisted token	TokenNotActive
Trade against a pool below the liquidity floor	PoolNoLiquidity
Create a vault without a credential	NotApproved
Register twice, or claim twice	AlreadyRegistered / AlreadyClaimed
Transfer the ManagerNFT	reverts – soulbound

One trade per block prevents multi-leg atomic sequences a real trader could not execute. **The hold time** stops instantaneous round trips against a momentary tick. **The minimum trade size** stops farming the trade counter with a hundred dust trades. And **the fee makes churn expensive by arithmetic rather than by rule**: the simulator never creates value, so round-trip trading only ever loses money. Ten pointless round trips cost 6% of capital, which means a participant who needs ten trades and has no view starts needing to make back 6% before they begin.

Asset listing is validated too. A token can join the challenge only if it is not the base token, its pool actually contains the base token paired with it, and the contract can price it at the configured window *right now*. Balances are whole cents, so an asset worth less than \$0.01 divides to zero and cannot be priced; rejecting it at listing turns that into a deployment error rather than a silent valuation bug that would quietly under-report a participant's portfolio for as long as they held it.

MEASURED ON-CHAIN – THE CREDENTIAL FLOW, END TO END

A fresh wallet registered at exactly \$100,000.00, traded 11 times, drove its portfolio to **\$216,789**, claimed, and had ManagerNFT #1 minted with factory approval granted atomically. That manager then created a vault, deposited into it, and executed a trade.

A second wallet registered at exactly \$100,000.00, traded 12 times, reached \$99,784.68, and was refused a credential by every route in the table above. No NFT was minted and the factory never reported the wallet eligible.

8.4 The credential

Passing mints a **ManagerNFT**: an ERC-721 that cannot be transferred, one per address per chain, enforced by the contract. Its metadata is generated on-chain as a base64-encoded SVG with no external dependency, recording the epoch the holder passed in and their final portfolio value, so it renders in any wallet without a server.

The credential is permanent unless revoked. It survives closing every vault, long inactivity, and losing money. It does not survive `revoke()`, which burns the token and, with a factory ban, blocks new vault creation — existing vaults keep running and their depositors keep every exit route.

Failure handling. The mint and the factory approval are best-effort by design, so a downstream misconfiguration cannot undo a legitimately passed challenge. Their failures are not silent: each emits `CredentialEffectFailed`, and *anyone* can call `retryClaimEffects(address)` to re-run them. That retry

is permissionless on purpose — it can only ever grant a credential to an address whose `claimed` flag is already set, which is reachable only through the full gate check. Without it, an outage at the wrong moment would leave a passed participant permanently stuck.

8.5 One credential, every chain

Version 2 runs on two networks, and a manager should not have to double a virtual portfolio twice. The credential contract is deployed with CREATE2 from a chain-agnostic constructor, so it sits at the same address, `0x04DbC537...F98c6a7`, on MegaETH testnet and Base Sepolia, and will on any chain added from the same build. Token IDs are global, the origin chain ID times one million plus a sequence number, so the first credential earned on MegaETH is #6343000001 wherever it appears, and `originChainOf()` and `isMirror()` read its provenance straight off the ID.

No messaging layer spans the two testnets, so a credential crosses by **attestation**. An off-chain service reads the credential on the chain where it was earned, never from another mirror, and signs an EIP-712 message binding the holder, the token ID, the holder's current nonce on the target chain, and a deadline thirty minutes out. The holder submits it to `mintMirror()` on the target chain and pays the gas. The signature is single-use and cannot be redirected to another address, and the service simulates the exact call before releasing it. `VaultFactory.createVault()` approves any credentialled wallet at tier 1 on first use, so a mirror is all a manager needs to launch vaults on the second network. Tiers and approvals stay per chain.

WHAT THE ATTESTATION KEY CAN AND CANNOT DO

It can create a mirror of a credential whose ID names another chain, and remove a mirror whose origin is gone. It cannot mint a native credential, touch a credential on the chain it was earned on, reuse a signature, or mirror a credential back onto a chain where the owner revoked it: `revoke()` sets a block there that only an explicit owner call lifts. The residual risk is a compromised key minting mirrors for wallets that never passed, which could then launch vaults on the chains holding those mirrors. Depositors in such a vault keep every protection in \$5. The owner can rotate the key, or disable mirroring by setting it to zero.

The single-chain credential contract MegaETH used before version 2 is retired. Its one live credential was re-issued on the universal contract as #6343000001, and MegaETH's challenge and vault factory were pointed at the new address.

8.6 Timekeeping, and one honest exception

Sub-second blocks. MegaETH produces blocks in about a second, which makes `block.timestamp` too coarse for the 60-second hold windows the contract enforces. It reads MegaETH's microsecond-precision system oracle at `0x6342...0002` when that address has code, and falls back to `block.timestamp` scaled to the same microsecond units when it does not. The same bytecode therefore works on any EVM chain without a deploy-time change, which also lets the test suite run on a plain Foundry environment, and is what let the challenge deploy to Base Sepolia unchanged.

THE ONE DISCRETIONARY PATH IN THE SYSTEM

The platform owner can certify an already-registered participant directly with `ownerForceClaimEligibility()`, bypassing the numeric gates. It exists for someone vetted another way — a manager with a verifiable off-chain record, for whom grinding out a paper-trading target proves nothing useful. It still requires prior registration, so a credential cannot be conjured for an address that never opted in, and it emits a `ForceClaimed` event naming the caller.

This is where discretion lives, and it is deliberately visible on-chain. The owner still **cannot** change the starting balance or the target, which are compile-time constants, and **cannot** re-enable spot pricing, because `setConfig()` rejects a TWAP window below 60 seconds.

8.7 What passing does not mean

It means someone doubled a virtual portfolio against real prices, over at least a week, without breaking any rule and without being able to manipulate the price they were marked against. That is a filter, not an endorsement. One passed challenge is not a track record, and TroveFi does not vouch for anyone's future results. What depositors get instead is the complete on-chain history of every vault: NAV per share since inception, time live, current positions, and every trade ever made.

Fee economics

One charge exists: a performance fee on profit above the high-water mark. There is no management fee, no deposit fee, no withdrawal fee, no subscription, no listing fee, no lockup, and no early-exit penalty anywhere in the protocol.

Version 2 adds no charge. Community buckets pay nothing to the platform or to their creator. The one new payment, the Autopilot tip, goes to whoever executes an order rather than to TroveFi, applies only to orders a holder chooses to place, and is capped in bytecode at 0.5% of an execution (\$7.7).

9.1 The schedule

PERFORMANCE FEE BY TIER — COMPILE-TIME CONSTANTS

TIER	MILESTONE	TRACK RECORD	MANAGER	PLATFORM	TOTAL	VAULTS
1	pass the challenge	any	0.5%	0.5%	1.0%	1
2	\$1M+ TVL	30% IRR, 30-day rolling	1.0%	0.5%	1.5%	2
3	\$10M+ TVL	50% IRR, 30-day rolling	2.0%	0.8%	2.8%	5
4	\$50M+ TVL	100% IRR, 6 months live	5.0%	1.0%	6.0%	unlimited

The platform's share is roughly half the total at tier 1 and a sixth at tier 4: as managers prove themselves, more of the fee goes to them. These rates cannot be changed by the manager, the platform, governance, or an upgrade. A vault's tier can be raised, moving it to a different row, but the rows are fixed in bytecode.

Tiers only go up. `upgradeTier()` rejects any tier not strictly higher than the current one, and there is no downgrade path. If a tier 3 manager's TVL falls back below \$10M, their existing vaults keep the tier 3 schedule — the tier reflects what was achieved, not what is currently true. The lever the platform does hold is `revokeManager()`, which bans new vault creation for cause without changing existing vaults or depositors' exit routes.

Tier is stored in two places, and they are not the same thing. The factory stores the *manager's* tier, governing how many vaults they may create and what tier new vaults start at. Each vault stores its *own* tier, which is what its fee schedule reads. Promoting a manager therefore does not re-price their existing vaults; that requires a separate `upgradeVaultTier()` call on a specific vault. This is deliberate — raising the fee on a vault people already deposited into is a change to their terms, and it should be an explicit, visible, per-vault transaction rather than a side effect.

There is no paid tier. You cannot buy a tier, subscribe to one, or pay for a fee upgrade. The only inputs are size and results, both read from the chain.

9.2 The high-water mark

Suppose a manager charged a flat cut of every up month. A vault that runs +30%, -25%, +30%, -25% forever ends roughly flat for depositors while paying the manager on every up leg. The manager gets rich on volatility; the depositor funds it. A high-water mark makes the fee depend on the vault's peak rather than its

recent direction: losses must be earned back before anything is charged again.

```
accruePerformanceFee() // permissionless
  nav = navPerShare()
  if nav <= hwmPerShare: return // charge nothing

  gain      = (nav - hwmPerShare) × totalSupply // -> base units
  managerFee = gain × managerBps / 10,000
  platformFee = gain × platformBps / 10,000

  if liquidBase < managerFee + platformFee: return // change nothing at all

  hwmPerShare = nav // the PRE-fee NAV
  accruedManagerFee += managerFee // reserved, not paid
  transfer(platformRecipient, platformFee) // paid immediately
```

Setting the mark to the pre-fee NAV means the fee taken does not itself become a new hurdle. Because the mark never decreases, no gain is ever charged twice.

WORKED EXAMPLE – A TIER 3 VAULT, 2.0% MANAGER + 0.8% PLATFORM

EVENT	NAV / SHARE	HWMFEE CHARGED
Vault opens	1.0000	1.0000none
Trades well	1.2000	1.0000pending
accruePerformanceFee()	1.1944	1.20002.8% of the 0.20 gain
Loses money	1.0500	1.2000none
Recovers to 1.15	1.1500	1.2000none — still under the mark
Recovers to 1.30	1.3000	1.2000pending
accruePerformanceFee()	1.2972	1.30002.8% of the 0.10 above 1.20

Between 1.05 and 1.20 the manager earned nothing at all. That stretch was recovering their own drawdown, and depositors do not pay for it. For a manager, a 30% drawdown means working for free until it is recovered — there is no reset and no catch-up provision. That is the mechanism that makes a performance fee mean something.

MEASURED ON-CHAIN – ACCRUAL ON A LIVE TIER 3 VAULT

```
profit realised      51.76  tUSD
manager fee (200 bps) 1.0353 tUSD
platform fee ( 80 bps) 0.4141 tUSD
hwmPerShare      1.00000 -> 1.00518
```

Exact to the raw unit. A second accrual called immediately afterwards correctly did nothing.

9.3 Why the two halves are paid differently

The platform's share transfers out at accrual. The manager's share is *reserved* into `accruedManagerFee` and stays inside the vault until `claimManagerFee()` is called. While it sits there it is excluded from NAV, so it does not inflate the share price; excluded from withdrawal liquidity, so it cannot be paid to a depositor by mistake; and off limits to the trading logic, so a trade that would push the base balance below it reverts with `TradeExceedsLiquid`.

That last protection is not cosmetic. Without it a trade could push the balance below the reserved fee, and every subsequent NAV read, withdrawal, and fee claim would revert on underflow. The vault would be permanently bricked.

The liquidity condition on accrual exists for the same reason. Profit can be entirely unrealised: a vault up 30% on an open position has a NAV well above its mark and may hold almost no base token. Reserving a fee it cannot cover would push the reserve above the actual balance and brick the vault, so the accrual waits — changing nothing, including the mark — until the manager sells back to base or new deposits arrive.

Accrual is permissionless for an incentive reason. A manager who could choose when to accrue could time it; a manager who had to accrue manually might simply forget. Making it open to anyone means the fee is a function of the vault's state rather than of anyone's decision, and it fails safe in both directions.

9.4 What a depositor actually pays

10,000 TUSD HELD THROUGH A 40% GAIN

	BUCKET	VAULT, TIER 1	VAULT, TIER 4
Entry cost	-35 (0.35%)	0	0
Gross value at +40%	13,951	14,000	14,000
Manager fee	0	-20	-200
Platform fee	0	-20	-40
Exit cost	-33 (0.24%)	0	0
Net proceeds	13,918	13,960	13,760
Net return	+39.2%	+39.6%	+37.6%

Bucket costs are DEX pool fees paid to Kumbaya's liquidity providers, not TroveFi charges. Vault deposits and withdrawals swap nothing and cost gas only; a vault's swap costs land indirectly, through NAV, when the manager trades.

AGAINST THE TRADITIONAL STRUCTURE

	HEDGE FUND	TROVEFI VAULT	TROVEFI BUCKET
Management fee	2% per year	none	none
Performance fee	20%	0.5%–5%	none
Platform fee	none	0.5%–1%	none
High-water mark	usually	always	n/a
Lockup	months to years	none	none
Minimum	often \$100k+	none	none
Custody	the fund holds it	you hold shares	you hold shares
Reporting	quarterly letter	every block	every block

9.5 The manager's side of the arithmetic

TVL	ANNUAL RETURN	TIER	MANAGER'S FEE
\$50,000	25%	1	\$62
\$250,000	25%	1	\$312
\$1,000,000	30%	2	\$3,000

TVL	ANNUAL RETURN	TIER	MANAGER'S FEE
\$10,000,000	50%	3	\$100,000
\$50,000,000	100%	4	\$2,500,000

The structure is deliberately unrewarding at small size, because that is the correct price for someone with one passed challenge and no record. Fees rise only after the evidence does, and a manager cannot start at 5%. The honest failure modes on the manager's side are worth naming: most vaults attract nothing, because a vault with no track record and no distribution has no reason to be deposited into; a drawdown the manager does not recover from means working for free indefinitely; and with no lockup, capital is never committed, so a strategy that needs stable AUM does not fit this product at all.

Security model

There is no insurance fund, no coverage pool, and no compensation scheme. The protections are structural: the contract does not have the functions that would let anyone take the money, and it does have the functions that let you leave.

10.1 Four guarantees

1. Nobody can withdraw your funds but you. No function transfers assets to the manager other than `claimManagerFee()`, which pays only a fee already reserved against profit above the high-water mark, and none transfers to the platform other than the platform's share of that same fee. No admin, no governance vote, and no upgrade can add one, because deployed vaults are clones of a fixed implementation with no proxy admin.

2. Nobody can freeze your funds. Withdrawals work in every state, including after permanent closure. If a vault is closed, or paused for more than seven days, `redeemInKind()` returns a pro-rata slice of every underlying token with no swap, no oracle read, no counterparty, and no permission required. On a community bucket that route is open at all times (§7.6). Two exceptions are documented rather than hidden: a curated bucket pause stops withdrawals (§11.3), and an owner cycling vault pauses can hold the seven-day clock back (§11.6).

3. Prices cannot be forged. Every valuation, share mint, fee accrual and slippage floor reads a time-weighted average from the pool's own observation buffer, thirty minutes for vaults and five for buckets. Nothing reads spot and no external feed is involved anywhere.

4. Share price cannot be inflated. The virtual-offset mint prices the first share at exactly par regardless of any balance sitting in the contract, so the classic first-deposit donation attack does not apply.

10.2 Attack surface and defences

THREAT MODEL

ATTACK

DEFENCE

Flash-loan price manipulation	TWAP only, never spot, everywhere
First-deposit share inflation	virtual-offset mint; the seed always prices at par
Dust residue re-pricing later depositors	the last holder out sweeps the full balance
Sandwiching a manager's trade	on-chain minimum-out floored at TWAP – 5%
A lying or swapped-out router	the vault credits its own measured balance delta
Manager draining value through trades	every trade must cross base into a whitelisted token
Manager spending the fee reserve	trades checked against balance minus accrued fee
Unbounded NAV gas growth	NAV walks the vault's own positions, capped at 64
Delisting erasing a position's value	valuation reads the pool reference, not the active flag
Delisting stranding a position	selling is always permitted, whitelisted or not
A V2 pair registered as a price source	the registry probes V3-only methods at listing
Duplicate asset double-counted in a bucket	rejected at configuration time

ATTACK	DEFENCE
A stale pool cache after a fee-tier change	cache invalidated when the tier changes
Rebalance appearing to create value	reverts on any gain beyond a rounding epsilon
Re-entrancy	<code>nonReentrant</code> on every state-changing entry point; funds pulled before minting
Fee accrual bricking the vault	accrual skipped entirely unless liquid base covers both fees
Credential minted on a manipulated price	300-second TWAP; spot unreachable by configuration
Farming the challenge trade counter	minimum size, minimum hold, one trade per block, 30 bps per leg
A banned manager re-approving themselves	the ban flag is separate from approval and survives the self-approval paths
Factory used as a confused deputy	privileged calls verify the target was deployed by this factory
Buying a community bucket's record with a donation	NAV counts accounted balances only, never <code>balanceOf()</code>
Pushing spot, then rebalancing into it for a record	gains over TWAP are divided out of the track record
One good day, real or manipulated, taking a bucket public	the last four checkpoints and the live NAV must all clear the bar
A creator changing the basket under holders	one-day notice on every reweight once anyone else holds shares
A stranger extending someone's lock with a dust deposit	a deposit that locks must mint to its caller
An Autopilot stop fired by a price spike	triggers read the bucket's TWAP-priced NAV
A keeper filling an order badly	every fill floored at the TWAP preview less the owner's tolerance
A mirrored credential replayed or redirected	EIP-712 signature binds holder, chain, nonce and a 30-minute deadline
The attestation key minting a native credential	a mirror's ID must name another chain

10.3 Admin powers

TroveFi is not fully decentralised. Admin keys exist and can do real things. One rule shapes the whole list: **no admin function can move user funds**. There is no rescue function, no emergency withdrawal, no sweep, and no upgrade path on a deployed vault that could add one. Everything below is power over configuration and state, never over custody.

VAULT OWNER – THE PLATFORM		
POWER	EFFECT	RESIDUAL RISK
<code>pauseVault()</code>	blocks deposits and new positions	withdrawals continue; in-kind opens after 7 days
<code>closeVault()</code>	terminal retirement	both exit routes stay open, in-kind immediately
<code>upgradeTier()</code>	raises the fee tier	raises the fee on an existing vault. Upgrade only, emits <code>TierUpgraded</code>
<code>setRegistry()</code>	changes which tokens are tradeable and how priced	the strongest lever here — a malicious registry could distort NAV, hence share minting and fee accrual
<code>setSwapRouter()</code>	changes which router executes trades	a bad router causes reverts, not silent underpayment: the vault requires its own measured delta to meet the TWAP floor
<code>setTwapPeriod()</code>	adjusts the window	bounded to 60 s – 24 h

The factory owner can additionally approve or revoke managers, set tiers, change the implementation used for *future* vaults, migrate registries, set the default router, and change the platform fee recipient. Three details matter:

- **setImplementation()** does not affect existing vaults. Clones are immutable once deployed, which is why fixes do not propagate backwards — see §11.1.
- **approveManager()** is the bypass. The platform can approve a manager without the challenge. It is how a manually vetted manager gets in, and it is also the path someone would use to approve themselves. It emits `ManagerApproved`.
- **Revocation is deliberately sticky.** The ban flag is separate from the approval flag, because otherwise a banned manager still holding a challenge-earned credential could undo the ban by calling `createVault()` again — the self-approval path keys off "is this address currently unapproved". The ban survives until an explicit `approveManager()` lifts it.

BUCKET FACTORY OWNER, NEW IN VERSION 2

POWER	EFFECT	RESIDUAL RISK
<code>setBucketPolicy()</code>	rebalance, slippage, notice, checkpoint, lock and composition rules for every community bucket, immediately	changes the terms of buckets already held; within hard bounds, but the reweight notice can go to zero (§11.8)
<code>setGraduationPolicy()</code>	the bar for going public	bounded only by needing one recent checkpoint
<code>setDepositsPaused()</code>	stops new deposits on every community bucket	never touches withdrawals or in-kind redemption
<code>setSwapRouter()</code>	the router every bucket swaps through	buckets credit their measured delta against a TWAP floor, so a bad router reverts
<code>delist()</code>	makes a bucket private and bans it from graduating	holders keep every exit
<code>releaseLocks()</code>	voids every lock on one bucket, permanently	can only free holders
<code>skim()</code>	recovers tokens sent to a bucket outside its accounting	cannot touch an accounted balance

The Autopilot owner can change the keeper tip within its 0.5% cap, and nothing else: not cancel, redirect or execute out of turn anyone's order. The credential owner can set the minter, revoke a credential on a given chain, block or unblock mirroring for a holder, and rotate or disable the attestation key (§8.5). A community bucket's creator is not an admin in this sense; they can deposit while it is private, propose composition changes that wait out the notice, and lock their own shares.

The strongest protection against all of it is `redeemInKind()`, which reads no registry, no oracle and no router. On a community bucket it is always open. On a vault it opens automatically if the vault is closed or paused for a week, subject to the pause-cycling gap in §11.6, and closing opens it permanently. Whatever an admin does to configuration, that path still returns a proportional share of the actual tokens.

GOVERNANCE

Testnet ownership is a single development wallet on both networks. A mainnet deployment should move these keys to a multisig with a timelock. That has not happened, and it matters more now that community bucket policy reaches every existing bucket at once.

10.4 What has been verified, and what has not

FOUNDRY SUITE 329 / 329 27 suites: 3 invariant, 8 adversarial, 1 gas	LIVE VERIFICATION 2 chains real transactions, not forks	FRONTEND ROUTES 23 / 23 per network, production build, real wallet	THIRD-PARTY AUDIT none
FORMAL VERIFICATION none invariants are property tests, not proofs	BUG BOUNTY none		

Internal review covered access control, share accounting, oracle usage, re-entrancy, fee logic, gas bounds, upgrade paths and failure modes, producing a written findings document with severities and remediations. Several defences described in this paper came out of it: the delisting behaviour that erased NAV, the closed-vault withdrawal freeze, the unbounded NAV loop, the factory confused-deputy paths, the router self-report, and the fee reserve being spendable by trades. All are fixed, and a dedicated test file re-tests every finding.

Adversarial review, 13 September 2026. A second internal pass set out to steal funds or brick contracts rather than read them: a dedicated suite of exploit tests, a fresh deployment on MegaETH testnet isolated from the live one and attacked with real pool trades, and a static scanner over every contract. It found no way for an outside attacker to take funds or create value; first-deposit inflation, donations into held tokens, the phantom performance fee, and minting against a TWAP that disagrees with spot were each attacked directly and held. It did find the pause-cycling gap (\$11.6), a deposit path that credits the amount requested rather than received, which matters only for a fee-on-transfer base token, and a one-unit rounding difference reachable only in an emptied vault holding trapped assets.

Live verification exercised every subsystem with real transactions: bucket round trips and full exits on all five, a rebalance, deposits matching `previewDeposit()` exactly, fee accrual exact to the raw unit, tier upgrades on both factory and vault, in-kind redemption on a closed vault, the complete credential flow from registration to vault creation, every documented challenge rejection, and oracle correctness across all nineteen assets against an independent off-chain port of the tick math.

Version 2 was verified the same way on both networks: all 55 Base Sepolia addresses checked for code, a credential carried from MegaETH and used alone to launch a vault on Base Sepolia, community buckets created, seeded, deposited into and withdrawn from, and a keeper run against live orders in which a DCA buy and a stop-loss filled with every balance reconciling exactly. The runtime bytecode deployed for the version 2 contracts matches a fresh build of the repository on both chains, compiler metadata aside.

What that is not. Every review so far has been by people with an interest in the outcome, which is exactly the limitation an external audit exists to remove. An audit would not make the code correct; it would make the claim credible to someone with no reason to take our word for it, and it would find what the authors cannot see. If you are evaluating this for real money on a mainnet deployment, the absence of one should weigh heavily. And the version 2 contracts, `CustomBucket`, `BucketFactory`, `BucketAutopilot` and the cross-chain additions to `ManagerNFT`, were written after the adversarial review and have not had one.

10.5 Where the real risk is

Managers losing money is the largest risk by a wide margin, and no contract prevents it. A community bucket's equivalent is simpler: it holds what it holds, and a short record in a rising market clears the bar easily. After that: the contracts are unaudited, and the version 2 contracts are the least reviewed; admin keys

can affect valuation and community bucket terms even though they cannot move funds; mirrored credentials rest on a trusted attestation key; three known defects remain open; TWAP lags in a sharp move; and old clones keep old bugs, which is a real consequence of immutability rather than an oversight.

Known limitations

Things that are wrong, incomplete, or imperfect, written down rather than left for the reader to find.

11.1 Old vault clones carry the share-accounting bug

Affects vaults created before 29 August 2026. The original `TroveFiVault` minted 1,000 real dead shares and floored the payout in `withdraw()` — the same combination that broke BLUE v1 (§3.2). A vault emptied by its last holder is left with one base unit stranded behind unredeemable shares, pushing NAV per share to roughly 1e9 times par. The next deposit then triggers a performance fee on profit that never existed.

```
reproduced on a throwaway vault with real funds
seed 1,000 tUSD          navPerShare 1.000000, supply 1000.000000
full exit by the holder   totalAssets 1 raw unit, supply 1000 dead shares
                          navPerShare 1e27 (1,000,000,000× par)
next deposit 1,000 tUSD   previewDeposit 1e12 share-units, par is 1e21
accruePerformanceFee()    platform 4.999999 + manager 4.999999
victim redeemable         1000.000000 -> 990.000002 tUSD
NET                        -9.999998 tUSD = 0.99% of a 1,000 deposit
```

No attacker is required — this is the ordinary lifecycle. A manager could also trigger it deliberately by seeding and fully exiting their own vault before taking deposits, then collecting the manager share. Worst case is tier 4's 600 bps, so **6.00% of the next deposit**. Principal beyond the fee is safe in all cases, and ownership and redeemability stay correct.

Status: fixed in the implementation and deployed on 29 August 2026. Vaults created after that date are unaffected. Existing clones keep the old bytecode, because vaults are not upgradeable. The operational guidance is short: *do not let an older vault reach zero supply*. A vault that has already emptied displays the symptom plainly — a NAV per share around 1,000,000,000.

11.2 No performance-fee equalisation for new depositors

Affects every vault. A vault has one `hwmPerShare` for the whole pool, not one per holder. Deposit while NAV sits above the mark and you are in the pool when the fee on that pre-existing gain is charged, so you pay a share of profit earned before you arrived. Measured live: a 2,000 tUSD deposit followed by an accrual — which anyone can trigger — cost the new depositor 5.39 tUSD, or 0.2695%.

This is the classic pooled-fund equalisation problem rather than a coding error, but it is real money and it is not currently warned about at deposit time. **Mitigation today:** compare NAV per share against the high-water mark before depositing; if NAV is at or below the mark it does not apply. **Proper fix:** per-holder marks or an equalisation credit issued at deposit — both changes to the share model rather than parameter tweaks.

11.3 Curated bucket pauses stop withdrawals

Affects the five curated buckets, not community buckets. A vault pause leaves withdrawals working and opens in-kind redemption after seven days. A bucket pause halts **both** deposits and withdrawals, and buckets have no in-kind redemption path at all, so a paused bucket is a harder stop than a paused vault with no automatic escape after a delay. This asymmetry is not intentional design so much as the bucket contract predating the

vault's exit guarantees. There is no in-contract mitigation; the retired BLUE v1 was deliberately left unpaused for exactly this reason. Community buckets were written with the lesson applied: their exits cannot be paused and in-kind redemption is always open (§7.6).

11.4 Depositors pay for the manager's position count

The gas curve in §4.4 is bounded and safe, with no block-limit or denial-of-service risk anywhere on it. It is not obviously *reasonable*, though: the manager chooses the position count, and every depositor pays for that choice on every transaction, forever, without being able to opt out or see the charge coming. The cap stays at 64 for now — lowering it to 32 would halve the worst case but raises a migration question, because an existing vault already above a new cap could not open positions until it unwound. Documented rather than silently changed.

11.5 Vault withdrawals depend on manager liquidity

Working as designed, listed because it surprises people. Live example: a vault holding 11,093 tUSD of assets but only 5,819 in liquid base could honour 52% of its value in withdrawals; a holder of more than half that vault could not exit without the manager unwinding first. The manager can sell in every state, so nothing structurally blocks the unwind, and in-kind redemption opens on close or a seven-day pause — but in the normal live case, exiting a fully deployed vault requires the manager to act. The vault page shows liquid base next to total assets for exactly this reason.

11.6 The owner can keep a vault's in-kind exit shut by cycling pauses

Affects every vault. Found by the adversarial review, rated medium, open. `unpauseVault()` clears the pause timestamp and `pauseVault()` sets a new one, so an owner who pauses, unpauses and pauses again inside every seven days restarts the clock indefinitely. If the manager also stays in positions, `withdraw()` fails for liquidity and `redeemInKind()` never opens. It needs the platform key used against depositors and a passive manager, and it defeats a guarantee the contract explicitly makes, which is why it is here. A Foundry test reproduces it with funds held past sixty days. Closing a vault is unaffected: it opens in-kind redemption immediately and permanently. Fix: anchor the clock to the first pause, or to cumulative time paused. Community buckets have no clock to reset.

RESOLVED SINCE VERSION 1

Version 1 listed here that the landing page advertised vault types the contract does not implement: DeFi strategy vaults (PT/YT, lending, staking), user-created buckets, and 48-hour "flash alpha" vaults. That grid has been replaced with the four things that exist, and user-created buckets now exist as §7 describes.

11.7 Testnet infrastructure gaps

Faucet rate limits are per-instance memory unless a KV store is configured — the proof-of-work, HMAC signing, single-use challenges and 24-hour cooldown each work and have been verified, but on serverless the per-IP limit resets on cold starts. The wallet cooldown does not, because it is computed from the faucet's own on-chain transfer logs. Points count MegaETH curated-bucket activity only (§12). Testnet pools hold 240,000 tUSD each, which is realistic for normal sizes but shallow for large trades, and all of that liquidity was team-seeded, so there is no competition on the other side of any trade.

11.8 Community bucket rules can change under existing buckets

Affects every community bucket. Buckets read their policy from the factory on every call, so a change reaches all of them at once. Within the factory's hard bounds (§7.1) the owner could cut the reweight notice to zero, widen the per-swap allowance to 10%, change how readily buckets rebalance, or lower the graduation bar, which has no floor beyond needing one recent checkpoint. None of that moves funds, `redeemInKind()` reads none of it, and each change emits an event, but it does change the terms of buckets people already hold, with no timelock.

11.9 Autopilot depends on keepers, and TWAP lags

Affects Autopilot orders. An order runs when someone executes it. TroveFi runs a keeper and the tip pays anyone else to, but nothing guarantees one is running at the moment a condition is met; escrowed shares wait until their owner cancels. Triggers read TWAP-priced NAV, which stops a spike from firing a stop and also means a stop fires minutes behind a sharp fall, at the price then available. Below the owner's slippage floor, execution reverts and the order waits. Escrowed cost basis is pooled inside the contract, so a cancelled order can return shares whose displayed cost basis differs slightly from what went in; balances are unaffected.

11.10 Cross-chain credentials trust an attestation key

Affects mirrored manager credentials. A compromised attestation key could mint mirrors for wallets that never passed the challenge, and those wallets could launch vaults on the chains holding the mirrors (§8.5). Revoking a credential on one chain does not remove its mirrors elsewhere, and revoking a credential does not revoke a factory approval already granted; both need an explicit call per chain. And because both networks were deployed from one wallet, several Base Sepolia addresses equal unrelated MegaETH contracts, so an address means nothing without its chain ID.

11.11 Further gaps

In-kind redemption's *asset* leg has not run on-chain against a multi-position vault (§5.4). And there is no third-party audit, no formal verification, no bug bounty, no mainnet deployment, and no independent economic or game-theoretic review of the fee model, tier thresholds, manager incentives, or the community bucket graduation bar.

Participation record

Points measure who actually used the protocol during the testnet. They are a record, not a promise.

A wallet joins by signing one message — free, no transaction, no gas, and no permission to move anything; the signature exists so nobody else can claim points for that address. On-chain awards have no claim button and nothing to submit: the server reads MegaETH directly and credits what the logs prove, with every entry in the activity feed linking to the transaction that earned it. Points cannot be faked because they do not come from what a browser reports.

AWARD STRUCTURE

CATEGORY	EXAMPLES	POINTS
Daily, resets 00:00 UTC	faucet check-in (+2 per consecutive day, capped +20; every 7th day +60); bucket deposit; challenge trade; withdrawal	10 / 15 / 5 / 5
Weekly, resets Monday	check in on 5 separate days; 3 bucket deposits; 15 challenge trades	100 / 75 / 100
Milestones, once per wallet	first deposit; first withdrawal; pass the challenge; launch a vault; link an X account	50 / 50 / 2,000 / 5,000 / 25
Quizzes, graded server-side	four quizzes on buckets, vaults, the challenge, and points	50–100 each

A maxed-out day is about 205 points; a perfect first week of check-ins is 172. Caps exist because none of these actions costs much on a testnet — without them the top of the leaderboard would belong to whoever wrote the best loop, which measures nothing worth measuring. The full manager path is worth 7,000, roughly a month of maxed-out daily activity, because it is the hardest thing in the product and is priced like it.

Quiz grading happens on the server and answers are never sent to the browser, so there is nothing to read out of the page. A failed attempt reports a score but not which questions were missed — otherwise a five-question quiz becomes a three-guess quiz on the retry. Retries are limited to five an hour per quiz.

Referrals pay 10% of an invitee's earnings on top, indefinitely, never deducted from the invitee. A referrer is set once and cannot be changed; self-referral and mutual referral are blocked; the bonus does not cascade; and fractional awards accrue rather than being rounded away. Invite capacity starts at 10 and unlocks to a hard ceiling of 250 — but the last hundred slots require a launched vault, which requires a passed challenge, so farming invites alone leaves a wallet at ten.

Durability. Every award is written to an append-only journal before it touches a balance, and balances are recomputable from it. On-chain points are recomputable twice over: from the journal, and from MegaETH itself. A wallet joining today is credited for its last seven days of on-chain activity and nothing older.

Honest limit. Some social tasks — following, reposting, joining a chat — cannot be verified, because X does not expose the follow graph to an app on the free tier and chat membership is not ours to read. Those are self-confirmed behind a short server-timed gate and pay accordingly; the amounts are small precisely because the honour system is what backs them. The on-chain half of the program has no such gap.

Scope. Points read MegaETH only, and count bucket activity in the five curated buckets only. Base Sepolia and community buckets earn nothing yet. Counting them needs per-chain records in the ledger, a change to make deliberately rather than patch in.

TroveFi has not announced what, if anything, points will be used for, and nothing on the points page should be read as a commitment.

Roadmap

Four lists, no dates. Dates on a roadmap are usually decoration.

13.1 Shipped in version 2

User-created buckets, listed as "asked for" in version 1, exist: permissionless deployment, a listing policy written as on-chain numbers rather than a review, and a track record built to be hard to fake (§7). **Autopilot** standing orders and **backtesting** came with them. **A second network**: Base Sepolia runs the same contracts as MegaETH testnet, which puts the protocol's code on the network it names as its mainnet target. **A manager credential that works everywhere**, at one address with attested mirrors (§8.5). **Bucket exit guarantees**, for community buckets: exits cannot be paused and in-kind redemption is always open. **Landing page claims**: the homepage no longer advertises vault types the contract lacks.

13.2 Being fixed

These are known defects with agreed fixes, specified in §11. **Pause cycling**: anchor a vault's in-kind clock to the first pause, or to cumulative time paused, so an unpauses cannot reset it. **Fee equalisation**: the cheap mitigation is a warning at deposit time when NAV sits above the mark; the correct fix changes the share model. **Curated bucket exit guarantees**: vaults and community buckets have both protections, and the curated five should too. **Old vault clones**: the options are a migration path or a standing operational rule, and neither is decided. **Governance keys**: mainnet needs a multisig with a timelock, more so now that community bucket policy reaches every existing bucket at once.

13.3 Being built

A third-party audit is the single most important item on this page for anyone considering real money, and nothing about a mainnet deployment should be taken seriously before it happens. It should cover the version 2 contracts, which have not yet had even an internal adversarial pass. **Private vaults** would let a manager restrict deposits to an allowlist; there is no contract, no finalised design, and no timeline, and what it needs is deposit-side access control plus a decision about whether the vault stays publicly readable, which it should. **Mainnet** targets Base as the reference chain, a choice that has already shaped the protocol: every bucket constituent was selected against whether the asset actually exists there, with gaps recorded per asset rather than glossed over. The deploy script records a mainnet starting point for the community bucket bar: a 30-day record, seven daily checkpoints inside eight days, +10%, and a creator lock with 90 days left.

13.4 Asked for, no commitment

More strategy primitives: lending, staking, LP positions and yield tokens each need new contract surface, new whitelisting rules, and new valuation logic, because NAV currently assumes every position is a plain token balance priced by one pool. Nothing is in progress. **More curated buckets**: the L2 index was shelved, and a memecoin bucket of Base-native names is the most obvious theme that would pass the filters; neither set of assets is in the registry, so neither can be built as a community bucket either. **More listed assets**: the registry now bounds what users can build as well as what managers can trade. **Points across networks and community buckets** need per-chain records in the ledger. **A token** is referenced in the tier 4 perk list as a governance weight; there is no contract, no supply, no distribution plan and no date. **An analytics API** would be a convenience layer over data that is already on-chain. **A bug bounty** does not exist today.

The being-fixed list is the honest one. Those are real defects, found in review, written down, and not yet closed. A protocol that publishes that list is easier to evaluate than one that does not, which is why it is here rather than in a private document. The last two are intentions; treat them as such.

Deployment reference

CHAINS 6343 · 84532 MegaETH testnet, Base Sepolia	BLOCK TIME ~1 s · 2 s	BASE TOKEN tUSD 6 decimals, one per chain	LISTED ASSETS 19 per chain, fee tier 3000
POOL LIQUIDITY 240k tUSD per pool	CURATED BUCKETS 5 per chain; BLUE v1 retired		

CORE CONTRACTS, MEGAETH TESTNET

CONTRACT	ADDRESS	PURPOSE
Base token (tUSD)	0xAd67E44f66A54f7E5C287fb271f4bA16744B Fa15	deposit token, 6 decimals
VaultFactory	0x7540eB4C...	deploys vault clones for approved managers
TokenRegistry	0xD90ED7FD...	platform whitelist, 19 active tokens
BucketRegistry	0x6aA7D4cf...	on-chain bucket discovery
ManagerChallenge	0xde3145C8...	the credential gate
ManagerNFT	0x04DbC53727879FE27F40EdA3AcAb556B1F98 c6a7	soulbound manager credential, same address on every chain
BucketFactory	0xf54500Aad2c70411C9ec01Fc64182EC2b7d9 1D48	community bucket deployment, policy and graduation
BucketAutopilot	0x8D2d395F473eC7003E00C8b74e6788096F36 3108	standing orders on community buckets
Multicall3	0xcA11bde05977b3631167028862bE2a173976 CA11	standard batch reader

Truncated entries are truncated deliberately: the canonical source is the deployment record and each contract's explorer page, not this document. Verify any address before interacting with it. The single-chain ManagerNFT MegaETH used before version 2, `0x8CD78CA3...`, is retired; its one credential was re-issued on the universal contract as #6343000001.

CURATED BUCKET CONTROLLERS, MEGAETH TESTNET

TICKER	NAME	CONTROLLER
BLUE	Crypto Blue Chips	0xd56cD450feA74f141Fe2c335aAd246D5549c F811
AIML	AI & Compute	0xb54070DB3c80c976ff5ff8FfE5E59B380c6D 5A08
DEFI	DeFi Blue Chips	0x20582Bac51Bf2639f13929d9B89df15c1be6 61df
RWA	Real World Assets	0x225f455A91d0EB2e34588FFDA54f28d38202 8A82
PAYFI	PayFi & Settlement	0xC2d8bD8371F8f87Bcc2C1300160e86BF27eb 16aF
BLUE v1	retired 2026-08-29	0x4aF5161a918f11493d686C46cb5ce6d008fe Ae0a

BASE SEPOLIA, DEPLOYED 15 SEPTEMBER 2026 FROM BLOCK 46853049

CONTRACT	ADDRESS	PURPOSE
Base token (tUSD)	0xb762342215F0C06D9d575BEe74B5dC413A1DdEBC	deposit token, 6 decimals
VaultFactory	0x53440097518AC95a9D0c75a79eCb546C592d9aCe	vault clones for approved managers
TokenRegistry	0x87C80272E12Ab640D96FBE40C4e3B04187d6180	platform whitelist, 19 active tokens
ManagerChallenge	0x7540eB4CFE93C30FF3B94650C007aBf49d7eDFcc	the credential gate
BucketFactory	0x225f455A91d0EB2e34588FFDA54f28d382028A82	community bucket deployment, policy and graduation
BucketAutopilot	0x3407030B7B2F5639fE654DAe3b228f8c36425f71	standing orders on community buckets
BLUE controller	0xC088d4244E852158b5546a70f1A2C3F442e3ecCa	Crypto Blue Chips
AIML controller	0x7Be6F156D44B0c88B13e36338507242db2D8DD2E	AI & Compute
DEFI controller	0xA092fD7E1BADd7A35b16dD8c1229AfCB834762bd	DeFi Blue Chips
RWA controller	0x5855A3663BbB4ca2500AFC2E02FF51e8E6E24fb5	Real World Assets
PAYFI controller	0xD5522D1ad00FaEb5c5Ffe603bc8C88FDaE5d4393	PayFi & Settlement

Swaps and prices on Base Sepolia go through the canonical Uniswap V3 deployment. The complete record, including every asset and pool, is [solidity/deployments/84532.json](#). Several of these addresses are identical to unrelated MegaETH contracts, because both networks were deployed from one wallet; always pair an address with its chain ID.

SOURCE FILES

FILE	LINESWHAT IT IS
CustomBucket.sol	1,044community bucket: book, rebalance, reweight, track record, locks
BlueBucketController.sol	877curated buckets: weights, rebalancing, slippage guards
TroveFiVault.sol	762share accounting, trading, fees, redemption
ManagerChallenge.sol	681the credential gate
BucketFactory.sol	539community bucket deployment, policy, graduation, previews
ManagerNFT.sol	362soulbound credential, cross-chain mirrors, on-chain SVG metadata
BucketAutopilot.sol	331DCA, stop-loss, take-profit, trailing stop
VaultFactory.sol	312vault deployment and manager tiers
KumbayaTwapOracle.sol	211TWAP pricing, adapted from Uniswap's OracleLibrary
TokenRegistry.sol	198asset whitelist and pool map
BucketRegistry.sol	156curated bucket discovery
BucketShareToken.sol	67curated bucket share ERC-20

FILE	LINESWHAT IT IS
PnlMath.sol	31shared PnL arithmetic
SafeDecimals.sol	25safe decimals() reads
InflationGuard.sol	18deprecated constants, kept for ABI stability

14.1 Reading a vault from the chain

The interface is a viewer, not a source of truth. Everything it shows is readable directly: `totalAssets()`, `navPerShare()`, `getPositions()`, `previewDeposit()`, `previewWithdraw()`, `previewRedeemInKind()`, `getUserPnl()`, `hwmPerShare()`, `accruedManagerFee()`, `currentTotalFeeBps()`, `inKindRedemptionOpen()`, `timeUntilInKindRedemption()`, and `state()`. Curated buckets add `getCurrentWeightsAndDrift()`, `getUserApy()` and `guardsArmed()`. A community bucket exposes `getConstituents()`, `navPerShare()`, `getCheckpoints()`, `windfallFactor()`, `pendingReweight()` and `lockOf()`, and its factory serves `graduationStatus()`, `getPortfolio()` and the deposit and withdrawal previews. Autopilot orders are read with `ordersOf()` and `isReady()`.

Events worth indexing: `Deposited`, `Withdrawn`, `TradeExecuted` (carrying the *measured* amount received), `FeesAccrued`, `ManagerFeeClaimed`, `VaultStateChanged`, `HoldingOpened`, `HoldingClosed`, `RedeemedInKind`, `CostBasisTransferred`, `Rebalanced`, `VaultCreated`, `ManagerApproved`, `ManagerRevoked`, `ChallengePassed` and `ForceClaimed`. Version 2 adds `BucketCreated`, `BucketGraduated`, `BucketDelisted`, `BucketPolicyUpdated`, `CheckpointRecorded`, `WindfallExcluded`, `ReweightProposed`, `Locked`, `OrderExecuted` and `CredentialMirrored`.

TWO DEPRECATED VIEWS

`getLeaderboardRaw()` and `getGlobalStats()` are retained for ABI compatibility only. They cost a pool read per participant per token and will exceed node gas caps in the low hundreds of participants. New integrations should use `getLeaderboardRange(offset, limit)` and `getGlobalCounts()`.

14.2 What carries over to mainnet

The contract behaviour carries over exactly — share accounting, fee logic, oracle reads, access control and exit guarantees are the same bytecode. What does not carry over is the market. Testnet pools are held near real prices by a one-sided market-making bot that trades each pool's price toward live spot for the asset it represents, so the TWAP mechanism runs exactly as it would on mainnet and prices move when the real asset moves. But real pools have deeper liquidity, more competition, worse fills at size, and nobody holding a peg for you. A strategy that works on testnet has been tested against real price movement and real AMM mechanics — not against real adversaries.

TroveFi Protocol Whitepaper, version 2.0, September 2026. Compiled from the protocol documentation and the deployed contract sources. Every measured figure comes from the MegaETH testnet and Base Sepolia deployments described in §14. Version 2.0 adds §7, §8.5 and the version 2 material throughout, and supersedes version 1.0.

The contracts have not been audited by a third party. Testnet tokens have no value. Nothing in this document is an offer, a solicitation, investment advice, or a prediction of returns. Read §11 before forming a view.