

Mulyr Protocol

Whitepaper v1.0

Rule-based, risk-constrained capital allocation across DeFi strategies

Mulyr Foundation
2026

Abstract

Mulyr is a non-custodial capital allocation protocol for the Ethereum Virtual Machine. The protocol routes USDC deposits through a modular ERC-4626 vault into a set of yield-generating strategies deployed across existing DeFi lending protocols, applying deterministic, rule-based allocation logic under enforceable on-chain constraints. Mulyr does not implement a lending primitive. Its role is to structure how capital is distributed across external protocols, when it moves, and under what conditions exits are satisfied — without discretionary decision-making at any layer.

The core vault (CoreVault) implements a Diamond-lite proxy architecture with EIP-7201 namespaced storage and an immutable selector registry. Capital flow is governed by three independent layers: a hot/warm liquidity buffer managed by BufferManager, an inter-strategy allocation router (StrategyRouter), and a set of isolated adapter contracts each scoped to a single external protocol integration. Rebalancing is executed in controlled batches subject to pre-execution validation, oracle freshness checks, NAV delta bounds, and aggregate loss caps — and reverts on post-execution invariant violation.

The MTRY governance token is issued at a fixed supply of 300,000,000 tokens with no post-issuance inflation. Its price reference is defined as treasury value divided by circulating supply — the subset of tokens that are economically active — with aggregate unlock velocity governed on-chain by the GlobalUnlockController. Protocol fees are distributed deterministically: approximately 70% to treasury, 29% to operations, and 1% to a safety reserve.

1. Introduction

1.1 Motivation

DeFi lending markets — including Aave [1], Compound [2], Morpho Blue [3], Euler [4], and Fluid [5] — produce yield as a function of per-market utilisation dynamics. Supply rates diverge continuously across venues and change as borrow demand, liquidity, and incentive programs evolve. A position that is optimal at time t may have degraded materially by $t + \Delta$, yet monitoring and reallocating across fragmented venues does not scale for capital of meaningful size.

Two structural risks compound the allocation problem. First, concentration in a single protocol introduces avoidable tail risk: a smart contract vulnerability, an oracle failure, or a governance parameter change in one protocol can produce partial or total loss with no containment. Second, naive yield-chasing — moving capital continuously toward the highest instantaneous rate — incurs gas costs and slippage that erode net returns, while reacting to transient spikes generates churn without durable benefit.

Existing approaches to this problem fall into two categories. Yield aggregators operate through strategy contracts that execute migration logic at the discretion of off-chain strategists or governance votes, introducing principal-agent risk between the protocol and its depositors. Multi-strategy ERC-4626 wrappers distribute capital across a fixed set of underlying vaults without a dynamic rebalancing layer, providing diversification but no ongoing allocation management.

1.2 Introducing Multyr

This paper introduces Multyr, a non-custodial allocation layer that structures how capital is deployed across DeFi strategies under predefined, enforceable on-chain rules. The protocol separates capital entry, liquidity management, strategy routing, and protocol-level execution into four independent layers. Each layer operates within a defined scope; failures at one layer are contained without propagating to others.

Multyr makes no discretionary allocation decisions. Governance configures parameters — exposure limits, rebalancing thresholds, fee rates — within bounded ranges and subject to timelock delays. All day-to-day capital movement is governed by on-chain logic. This design eliminates the principal-agent risk that characterises discretionary allocation systems: the protocol cannot deviate from its configured rule set.

The protocol is designed for deployment across multiple EVM-compatible networks; the first deployment is on Arbitrum One (chain ID 42161). Each deployment is independent, with its own vault instances, strategy configurations, and governance parameters. Capital is not bridged between deployments.

2. Capital Allocation Model

2.1 Allocation Architecture

Multyr structures capital allocation across two independent layers. At the system layer, a StrategyRouter distributes capital across one or more Strategy Vaults according to a configured allocation mode. At the strategy layer, each Strategy Vault manages capital deployment across its set

of protocol adapters under its own constraint framework. The two layers share no state: the system layer does not observe intra-strategy adapter composition, and each strategy does not observe the composition of sibling strategies. This separation bounds the impact of a failure at either layer.

An allocator may enter the system at either layer. A deposit into an **Allocation Vault** (Multi-Strategy) delegates the full allocation hierarchy to the protocol: the StrategyRouter determines inter-strategy exposure, and each Strategy Vault manages intra-adapter distribution. A deposit directly into a **Strategy Vault** (Single-Strategy) fixes the strategy class and delegates only intra-adapter management. Both entry points share the same underlying vault mechanics and exit guarantees.

2.2 Allocation Modes

The StrategyRouter supports three allocation modes, configured per vault:

PRIORITY mode. Deposits are routed first to the highest-priority target (lowest priority index). Redemptions drain from the lowest-priority target first. This mode concentrates capital in a primary strategy and uses secondary strategies only when the primary reaches capacity or is unavailable.

WEIGHTED mode. Deposits are split according to configured target weights. Redemptions are drawn proportionally to current asset distribution across targets. This mode maintains a defined allocation profile across multiple strategies.

NONE mode. Automatic routing is disabled at the router level. Capital movement must be triggered explicitly. Used during bootstrap or when allocation is managed through a separate governance process.

2.3 Open-Ended vs Fixed-Term Vaults

CoreVault supports two operating modes, determined at deployment and immutable thereafter, controlled by the FixedMaturityModule state machine.

Open-Ended Vault. Deposits and exits operate continuously with no fixed term. Instant withdrawals against the hot buffer are available subject to a per-epoch rate cap. Queued withdrawals are settled at epoch boundaries. This mode is suited to capital requiring ongoing liquidity access.

Fixed-Term Vault. Capital is committed for a defined duration. Deposits are accepted during a funding window; the strategy activates at the start date; exits become available at maturity. If minimum funding is not reached by the funding deadline, depositors may reclaim capital at the pre-failure price-per-share through a permissionless refund path. All withdrawals are queue-based; instant exit is not available. This mode suits strategies whose underlying positions carry inherent exit latency, where continuous exit availability would create structural liquidity mismatches.

2.4 Constraint Enforcement

All allocation behaviour operates within a layered constraint framework enforced automatically by the system. Constraints are not optional guidelines; the protocol cannot act outside them regardless of governance intent.

Constraint	Description
Exposure limits	Hard caps on capital allocated to any single Strategy Vault or adapter
Liquidity constraints	Allocation sized to withdrawable capacity and market depth of each target

Buffer requirements	Minimum liquid reserve maintained for withdrawal readiness at all times
Execution thresholds	Minimum net benefit required before capital movement is triggered
Loss caps	Per-target and aggregate loss tolerances enforced during redemption batches
Oracle freshness	Price feeds validated against configurable staleness bounds before execution

The system implements an asymmetric control design: restricting or halting capital allocation requires lower authority and no timelock, while resuming or expanding allocation requires governance action through the full timelock cycle. This asymmetry prioritises capital protection over operational speed.

3. Vault Mechanics

3.1 ERC-4626 and Share Accounting

CoreVault is fully compliant with ERC-4626 [6]. Allocators deposit USDC and receive vault shares representing a proportional claim on total vault assets. The price-per-share (PPS) is defined as:

$$\text{PPS} = \text{totalAssets}() / \text{totalSupply}()$$

where `totalAssets()` aggregates all USDC held across: (i) the CoreVault hot balance, (ii) warm adapter positions managed by `BufferManager`, and (iii) all deployed strategy positions reported by `StrategyRouter`. PPS is non-decreasing under normal operation. The only mechanisms that may reduce it are force-exit penalties and performance-fee crystallisation, both strictly bounded by on-chain parameters in `GlobalConfig` (`IParamsProvider`).

CoreVault implements a Diamond-lite proxy architecture: a thin ERC-4626 shell that dispatches all economic operations to external module contracts via *delegatecall*. The fallback dispatcher reads `moduleOf[msg.sig]` from `CoreStorage`, validates the caller role against `roleOf[msg.sig]` using the immutable `SelectorRegistry` (91 selectors at deployment), and delegates to the appropriate module. State is partitioned into four EIP-7201 [7] namespaced slots, eliminating storage collisions across module upgrades:

Namespace	Contents
CoreStorage	Module routing table, role registry, system flags, component references
FeeStorage	Fee parameters, high-water mark for performance fee crystallisation
QueueStorage	FIFO withdrawal queue, epoch state, settlement scan cursors
FixedMaturityStorage	Fixed-Term state machine, funding window configuration, maturity lifecycle

Standard ERC-4626 `withdraw()` and `redeem()` always revert with `AsyncWithdrawalRequired`, maintaining compliance via `maxWithdraw()` and `maxRedeem()` returning zero. All exits are request-based through the queue protocol described in Section 3.3.

3.2 Buffer Management

`BufferManager` maintains a two-tier liquidity layer between CoreVault and deployed strategy positions. The design ensures withdrawal availability without continuously unwinding strategy positions, and eliminates idle capital in the buffer layer.

Hot buffer. USDC held directly in CoreVault — immediately accessible for withdrawal settlement without external protocol interaction. Funded by deposit inflows and warm buffer refills triggered on settlement demand.

Warm buffer. USDC deployed into short-duration, instantly-redeemable protocol positions that earn yield while remaining near-liquid. BufferManager holds no idle asset balance at rest: excess hot USDC is deployed to warm adapters; the hot buffer is refilled only when settlement demand requires it. Buffer targets are defined as a percentage of total vault assets and are governance-configurable parameters.

A warm NAV cache provides pricing for buffer operations. If the cache exceeds its maximum staleness bound, the deposit path enforces a refresh before proceeding. Deposits are rejected when warm NAV is invalid or stale; withdrawals are never blocked by buffer pricing failures.

3.3 Deposit and Withdrawal Flows

Deposit

On deposit, USDC is transferred to CoreVault, shares are minted at current PPS, and a deposit fee of 0.25% is applied. Capital is initially held in the hot buffer. LiquidityOpsModule periodically deploys surplus capital to strategies via StrategyRouter. Newly deposited capital is subject to a lock period, during which withdrawal eligibility is restricted.

Standard Withdrawal

An allocator calls requestClaim(immediate=false, shares), which reserves the specified shares and places the request into a FIFO queue. Settlement is executed at epoch boundaries by the keeper calling settleFeesAndProcessQueue(). The settlement algorithm sources liquidity from buffers first, then from strategy positions. All claims within a settlement batch use a single PPS snapshot taken at batch initiation, preventing intra-batch arbitrage. No second transaction from the allocator is required. A 0.25% withdrawal fee applies.

Instant Withdrawal

Open-Ended Vaults offer an instant settlement path via requestClaim(immediate=true). Settlement is atomic in the same transaction when three conditions hold simultaneously: the lock period has passed; the epoch cap has not been exhausted; and the hot buffer holds sufficient USDC. The epoch cap — expressed as basis points of total NAV per epoch — rate-limits instant exits and prevents coordinated buffer drainage. If any condition fails, the request falls back automatically to the standard queue without consuming epoch cap. A total fee of 1.25% applies (0.25% base plus 1.00% liquidity premium), pricing the immediate-exit option explicitly.

Force Withdrawal

forceWithdraw() is an emergency exit path available to any allocator. It bypasses the lock period and does not consume epoch cap. The allocator supplies a liquidity plan specifying which strategy positions to redeem; execution follows a deterministic waterfall from warm adapters to strategies in allocation order. A variable fee applies above the base withdrawal rate. Force withdrawal is not the standard exit path and is available under degraded or constrained system conditions.

Exit path	Open-Ended Vault	Fixed-Term Vault
Standard (queued)	Always available	Available at maturity

Instant	Available, subject to epoch cap	Not available
Force	Emergency only	Emergency only

3.4 NAV and Oracle Safety

Primary price data is sourced from Chainlink Aggregators [8] via `latestRoundData()`. Each feed is validated against configurable staleness bounds. An optional secondary oracle provides cross-validation: if the deviation between primary and secondary prices exceeds a configured threshold, the transaction reverts. A consistent asymmetric failure policy is enforced: deposits are blocked when pricing data is invalid or stale; withdrawals are never blocked by oracle failures. This ensures allocators retain exit capability under all oracle conditions.

Oracle condition	Deposits	Withdrawals	Strategy ops
Primary feed stale	Blocked	Allowed	Blocked
Secondary oracle down	Allowed	Allowed	Allowed
Both feeds stale	Blocked	Allowed	Blocked
Deviation exceeded	Blocked	Allowed	Blocked

4. Rebalancing

4.1 Design Principles

Rebalancing in Multyr is not continuous and not reactive to every market fluctuation. Capital is redistributed only when predefined conditions are met, in a three-phase controlled process subject to pre-execution validation, batch execution with best-effort semantics, and post-execution invariant verification. This design avoids churn — unnecessary capital movement that incurs execution cost without improving allocation — while maintaining allocation accuracy over time.

4.2 Three-Phase Execution

Phase 1 — Pre-execution Validation

Before any capital moves, the system validates the batch against a set of independent guardrails. If any check fails, the batch does not proceed:

Guardrail	Check
Cooldown	Minimum elapsed time since last rebalance must be satisfied
Batch size	Number of actions in the batch must not exceed the configured maximum
Adapter allowlist	All targets in the batch must be governance-approved adapters
Oracle freshness	Price feeds for all affected positions must be within staleness bounds
Plan validity	Rebalance plan must be current; expired plans are invalidated before execution

Phase 2 — Batch Execution

If Phase 1 passes, the StrategyRouter executes the batch on a best-effort basis. Adapters that fail health checks are skipped; individual adapter failures do not abort the remaining batch actions. This improves operational resilience: a single degraded integration does not block capital movement across healthy targets. Critical failures — not recoverable adapter errors — override best-effort behaviour and trigger full batch reversion.

Phase 3 — Post-execution Validation

After execution, the system validates that the resulting state remains within acceptable bounds. Post-execution checks include: NAV delta limit (total assets must not have shifted beyond a configured threshold during execution), aggregate loss cap on redemption batches, and per-target loss caps where configured. If any post-execution check fails, the entire batch reverts. Rebalancing is therefore constrained not only before execution but validated after: a batch that produces unexpected losses is rolled back regardless of pre-execution approval.

4.3 Cost-Benefit Gating

Beyond the structural guardrails in Phase 1, rebalancing at the strategy layer is subject to an additional cost-benefit gate. A rebalance is only prepared when the expected net benefit of moving capital — estimated yield improvement minus gas cost and slippage — is meaningfully positive. This gate prevents the system from executing economically inefficient moves that would reduce net returns for allocators.

A complementary hysteresis condition requires that the proposed allocation change exceeds a minimum movement threshold before a rebalance plan is prepared. This filters micro-adjustments that pass the cost-benefit gate on paper but generate unnecessary operational overhead. Both conditions must hold simultaneously for execution to proceed.

The system also applies a stability adjustment to yield observations: rather than responding to spot APY, the scoring engine incorporates an exponential moving average over historical yield data. An adapter with high but volatile yield is treated more conservatively than one with a lower but consistent yield. This dampening reduces rebalancing frequency in response to transient spikes that would not persist long enough to justify execution cost.

5. Strategy Architecture

5.1 Adapter Model

Adapters are the execution layer connecting a Strategy Vault to a specific external protocol. Each adapter is isolated and scoped to a single integration. Its responsibilities are: deploy capital into the protocol on instruction from the strategy, report position value back to the strategy, and recall capital on demand. Adapters have no authority to initiate capital movement independently; they serve as controlled execution targets within the strategy framework.

Adapter isolation limits the propagation of integration failures. A bug or failure in one adapter does not expose capital held in other adapters. The strategy layer aggregates position values from all adapters to compute its reported NAV; if an adapter reverts on a balance query, the system falls back to its last bookkeeping value rather than reverting the entire NAV computation.

The initial strategy deployment — USDC Lending — integrates with Aave V3, Compound III, Euler V2, Fluid, Morpho Blue, Dolomite, and Venus on Arbitrum One. Additional adapters may be added through governance via the adapter allowlist, subject to the standard timelock process.

5.2 Strategy Lifecycle

Each Strategy Vault and adapter is tracked through a health state maintained by the StrategyHealthRegistry. Health state determines whether a target is eligible to receive new capital:

State	Description	Behaviour
OK	Normal operation; all checks passing	New capital may be allocated
DEGRADED	Persistent failures or abnormal conditions detected	No new deposits; rebalancing drains position over time
BROKEN	Severe failure; recovery requires explicit governance action	Hard isolation; no capital movement until restored

The Guardian role can set DEGRADED or BROKEN without timelock, providing immediate isolation capability. Restoration to OK requires a governance action through the full timelock cycle. This asymmetry — fast to stop, controlled to restart — is consistent with the broader security model described in Section 7.

5.3 External Protocol Interaction

Each adapter interacts with its target protocol through a standardised ILendingAdapter interface. The strategy does not call external protocols directly; all external calls are mediated through adapter contracts. This indirection provides a consistent failure boundary: external protocol reverts are caught and handled at the adapter level, with failures counted toward the adapter's health state rather than propagating to the strategy.

Adapter deployment modes differ by protocol: most adapters use a pull model where the adapter calls safeTransferFrom to move USDC from the strategy; one adapter uses a push model where USDC is pre-transferred before the deposit call. Both modes are supported within the same strategy framework without requiring changes to strategy-layer logic.

6. Treasury Economics

6.1 Fee Model

All fees are calculated and applied on-chain by the FeeCollector module. Fee parameters are bounded by constants in GlobalConfig (IParamsProvider); no governance action can set a fee above the protocol cap. Changes to fee parameters require a submit/accept/revoke cycle subject to the paramMinDelay timelock.

Fee	Rate	Applied when
Deposit	0.25%	On each deposit into any vault
Standard withdrawal	0.25%	On queued withdrawal settlement
Instant withdrawal	1.25%	0.25% base + 1.00% liquidity premium for immediate exit

Force withdrawal	Variable	Emergency path; base fee plus strategy-dependent penalty
Management	0%	Not applied
Performance	≥6%, ≤50%	Applied to profits above high-water mark only; cap hardcoded in GlobalConfig (defaultMaxPerfRate = 5e17 WAD)
Swap (Li.Fi)	0.15%	Mulyr routing fee on pre-deposit asset swaps via Li.Fi; Li.Fi fees are additional

Performance fees apply only to positive returns and only above the high-water mark, crystallised at epoch boundaries via `endEpochCrystallize()`. They do not affect deposited principal. The 1.00% instant withdrawal premium prices the liquidity option explicitly, discouraging opportunistic instant exits that would destabilise the hot buffer.

6.2 Fee Distribution

Collected fees flow through the `FeeCollector` to a deterministic on-chain split with no manual intervention or off-chain routing:

Destination	Share	Range	Purpose
Treasury Safe	~70%	65–80%	Protocol-owned capital accumulation
Operations Safe	~29%	15–30%	Development, infrastructure, audits, partnerships
Safety Reserve	~1%	1–3%	Limited loss-absorption buffer

The Operations Safe (2/3 multisig) holds no user deposits, token allocations, or audit bounty funds. It receives its share of protocol fees and may be refilled from the Treasury for operational expenses. Distribution percentages are governance-adjustable within the stated ranges, subject to the 48-hour timelock.

The Safety Reserve is not a loss guarantee mechanism. Major loss events are bounded at the allocation layer through per-strategy and aggregate loss caps, which halt or revert capital deployment before losses compound. The Safety Reserve covers minor operational contingencies only and should not be treated as a substitute for per-allocator risk assessment.

6.3 Treasury-Backed Token Model

The MTRY token economic model is structured around a treasury-backed price reference framework. The reference is defined as:

$$\text{Price reference} = \text{Treasury Value} / \text{Circulating Supply}$$

The denominator is circulating supply — the subset of total supply that is economically active and not locked in vesting schedules or treasury reserves — not total supply. Circulating supply at TGE is capped at approximately 7% of total supply (21,000,000 MTRY). Effective circulating supply at launch is determined by the quantity deployed into initial liquidity pools and may be materially lower than this cap. All remaining supply is locked: `PreMTRYClaim` (40,000,000 MTRY, 6-month cliff post-TGE); `VestingManager` (111,000,000 MTRY, 3–12 month cliffs); `EcosystemEmissionController` (controlled annual emission); `Treasury Safe` (non-circulating protocol reserve).

At TGE, treasury capital is composed primarily of pre-token sale proceeds. The circulating NAV reference at launch therefore reflects initial capital formation rather than steady-state fee economics. The model's long-term validity is contingent on protocol fee generation exceeding operational costs, producing treasury growth independent of token issuance. This transition is publicly observable through on-chain Treasury Safe activity.

Treasury growth derives from three sources: retained protocol fees (approximately 70% of all fees collected); yield on treasury capital deployed into protocol strategies; and initial capital formation from the pre-token sale (hard cap 800,000 USDC, at 0.08 USDC per preMTRY converting at 1:4 to MTRY, implying an effective acquisition price of 0.02 USDC per MTRY against the circulating supply reference). Treasury value is variable and depends on protocol usage, allocation outcomes, and market conditions. The price reference does not constitute a floor price or a redemption right against treasury assets.

6.4 Supply Schedule

MTRY total supply is fixed at 300,000,000 tokens. Minting is disabled after initial issuance; no inflationary mechanism exists post-TGE. Allocation at issuance:

Recipient / Contract	Amount (MTRY)	% of Supply
Treasury Safe	75,000,000	25.00%
Liquidity Safe	45,000,000	15.00%
PreMTRYClaim pool	40,000,000	13.33%
VestingManager	111,000,000	37.00%
EcosystemEmissionController	29,000,000	9.67%

The 111,000,000 MTRY held by VestingManager are distributed across five time-gated buckets with linear post-cliff schedules:

Bucket	Amount (MTRY)	Cliff	Duration
Team	51,000,000	12 months	36 months
Strategic Standard	27,000,000	6 months	18 months
Strategic Extended	18,000,000	6 months	24 months
Advisors Standard	10,500,000	6 months	18 months
Advisors Short	4,500,000	3 months	18 months

GlobalUnlockController enforces a hard cap on aggregate vesting unlocks, constraining the total supply that may be released per month. This cap is independent of the ecosystem emission programme: the 29,000,000 MTRY held by EcosystemEmissionController is distributed for protocol growth and incentives at a controlled annual rate, substantially lower than the monthly vesting unlock ceiling. The two figures govern distinct token flows and must not be conflated.

7. Governance

7.1 Separation of Powers

Protocol governance operates through a three-entity on-chain model with no single-key admin and no off-chain voting. All parameter changes and protocol actions are executed through timelocked multisig processes, with an independent veto right and a guardian role for emergency intervention. The three entities — governance multisig, guardian, and veto — are held by independent parties, reducing the risk of unilateral action.

Entity	Address	Type	Authority
ROOT_TIMELOCK	0x295CFB45012b5f40814C4b74b50F871D3f072798	Timelock	Executes queued governance actions after delay
SAFE_GOV	0x70ef444799D6FBbE0865bA598Bee6795e064a326	3/5 multisig	Proposes and executes all governance actions
SAFE_GUARDIAN	0x7407E68a5553E948eed862f19fc6B292eb48d677	1/1 multisig	Emergency pause and strategy isolation; no timelock
SAFE_VETO	0xF0c1A33d6741EB2dc174a0977095587a0648f1D7	1/1 multisig	Cancels any pending governance action unconditionally
Treasury Safe	0x75f65F192A47e5F9EaAE057bE90f52e85fDE1314	3/5 multisig	Holds protocol-owned capital
Operations Safe	0x80C791925eC19FA5dDD978E9A357D4D24AC4EABc	2/3 multisig	Operational expenses only
Safety Reserve	0x8c85B299ab189F2F88776D21AA94057a457b4f2B	2/3 multisig	Limited loss-absorption reserve

7.2 Parameter Change Flow

All protocol changes follow a deterministic execution path:

```
SAFE_GOV → schedule() → ROOT_TIMELOCK (48h) → execute() → activation
```

Vault-level parameter changes are subject to an additional internal delay (`paramMinDelay`, minimum 48 hours in production) between timelock execution and parameter activation. From scheduling to effect, critical parameter changes therefore require a minimum of 96 hours — sufficient time for allocators to observe and react to any pending modification before it takes effect.

During the timelock window, `SAFE_VETO` may cancel any queued action. This veto right is unconditional, requires only a 1/1 threshold, and does not itself require governance approval. No scheduled change is irreversible before execution.

7.3 Emergency Procedures

`SAFE_GUARDIAN` provides immediate emergency capability without timelock. Guardian authority is strictly asymmetric by design. The Guardian can halt activity — pause deposits and/or withdrawals, set a strategy to `DEGRADED` or `BROKEN` — but cannot restore normal operation. Resumption always requires a full governance action through the timelock cycle. This asymmetry ensures that

emergency isolation is available instantly while recovery requires deliberate multi-party action.

The protocol follows a progressive immutability path. After initial deployment with normal timelocked governance, `freezeRouting()` permanently locks the module selector table, preventing further module changes. Subsequently, `prepareSeal()` followed by `sealFinalState()` permanently locks the full vault configuration. Each step is irreversible. The sealing mechanism allows the protocol to converge toward full immutability as confidence in the parameter set increases.

8. Security Considerations

8.1 Access Control

CoreVault implements a five-tier role-based access control system. Every externally callable selector is mapped to a required role in the immutable SelectorRegistry at deployment. The mapping is defined at compile time; each `setModule()` call validates role assignment against the registry before writing, preventing misrouting of owner-only functions to public access. No selector is reachable without an explicit, validated role assignment.

Role	Caller	Capabilities
ROLE_PUBLIC (0)	Any address	Deposits, withdrawal requests, force withdrawal, read views
ROLE_OWNER (1)	Timelock multisig	Module upgrades, fee parameters, component wiring, sealing
ROLE_GUARDIAN (2)	Guardian multisig	Emergency pause, strategy state changes
ROLE_OWNER_OR_GUARDIAN (3)	Either	Shared emergency actions requiring rapid response
ROLE_MODULE (4)	Contracts only	Cross-module calls within delegatecall execution context

Module authorization is enforced through an `isAuthorizedModule` mapping checked before every privileged share operation (mint, burn, transfer). Ownership transfers require a two-step process (`beginOwnerTransfer` / `acceptOwnerTransfer`), preventing accidental loss of administrative control. Reentrancy is prevented through custom flag-based guards at the vault level, module-level non-reentrant execution, and additional guards on all external call sites.

8.2 Protocol Invariants

The codebase enforces a set of formally specified invariants. The following are critical to the correctness of the allocation and accounting model:

Invariant	Statement
PPS monotonicity	Price-per-share is non-decreasing under normal operation; reduction is bounded by on-chain fee parameters only
Exit guarantee	All exit paths remain open regardless of vault state; <code>DegradedMode</code> blocks deposits but never blocks withdrawals

USDC conservation	Total USDC tracked by each strategy (idle plus deployed) equals totalAssets() at every state boundary
Fee cap enforcement	No governance action can set any fee above the hardcoded maximum in GlobalConfig
Epoch cap guard	Instant exits are rate-limited per epoch; cap is enforced atomically at settlement
Sealing finality	After sealFinalState(), the module routing table is permanently frozen with no further upgrade path

8.3 Testing and Verification

The codebase has undergone extensive internal verification prior to external audit engagement:

Methodology	Scope
Unit tests	Component-level validation across all module logic and boundary conditions
Integration tests	Cross-module interaction testing including settlement, rebalancing, and fee flows
Fork tests	End-to-end tests executed against live network state on Arbitrum One
Formal verification	Halmos [9] symbolic execution (bounded). Specs cover USDC conservation, RBAC guards, and queue FIFO semantics. Proof artifacts at github.com/Multyr/multyr-core/tree/main/halmos-core . Zero counterexamples.
Static analysis	Automated vulnerability scanning with Slither [10] and Aderyn

The codebase is licensed under the Business Source License 1.1 (BUSL-1.1) with automatic conversion to GPL-2.0-or-later four years after the first production mainnet deployment, following the model of Aave v3 [11]. Public Solidity interfaces are separately licensed under MIT to enable third-party integrations without BUSL restrictions.

9. Limitations

9.1 Aggregation Risk

Multyr reduces concentration risk through diversified allocation across multiple external protocols, but introduces aggregation risk: a vulnerability in CoreVault or StrategyRouter would affect all vault capital regardless of the underlying protocol distribution. Allocators should treat Multyr's smart contract risk as additive to, not substitutive of, underlying protocol risk.

This aggregation risk is bounded by several structural constraints. Adapter and strategy layer isolation means no shared state exists between layers: a failure in one adapter does not expose capital in others. The five-tier role system and immutable selector registry (91 selectors, compile-time assignment) prevent unauthorized dispatcher access. Three-layer reentrancy protection covers the vault, module, and external call boundaries. Post-execution invariant verification reverts entire batches on NAV delta violations or loss cap breaches. The progressive immutability path allows the protocol to converge toward permanent vault sealing as confidence in the parameter set increases.

These mitigations reduce the attack surface but do not eliminate it. The codebase has not yet undergone external audit at the time of this publication. Allocators should weight this accordingly.

9.2 Oracle Dependency

The protocol relies on external oracle infrastructure for NAV calculation, deposit gating, and strategy operation. The oracle failure policy (Section 3.4) ensures withdrawals remain available under all oracle conditions, but deposit and rebalancing operations are suspended when pricing data is invalid. Prolonged oracle outages would prevent new capital deployment and strategy rebalancing while existing positions remain accessible for exit.

9.3 Keeper Dependency

Settlement, rebalancing, and fee crystallisation are triggered by a keeper role. If keeper automation ceases, these operations must be executed manually by any address with the appropriate role. Queue settlement delays would affect allocators awaiting standard withdrawal completion; instant and force withdrawal paths remain available without keeper involvement. The protocol is designed to degrade gracefully rather than lock capital under keeper absence.

Acknowledgments

The authors thank the contributors to the Morpho Blue [3], Euler v2 [4], Aave v3 [1], and Compound [2] protocols, whose open design choices informed elements of the Multyr architecture. The ERC-4626 standard [6], EIP-7201 [7], and the Chainlink decentralized oracle network [8] are foundational dependencies of the protocol.

Disclaimer

This paper is for general information purposes only. It does not constitute investment advice or a recommendation or solicitation to buy or sell any investment and should not be used in the evaluation of the merits of making any investment decision. It should not be relied upon for accounting, legal, or tax advice or investment recommendations. The opinions reflected herein are subject to change without notice. DeFi protocols carry significant risks including smart contract vulnerabilities, liquidity risk, market risk, and regulatory risk. Multyr does not guarantee returns or capital preservation. Allocators are responsible for conducting their own due diligence and for compliance with applicable laws in their jurisdiction.

References

- [1] Aave Team. Aave Protocol Whitepaper v2.0. December 2020.
- [2] Leshner, R., Hayes, G. Compound: The Money Market Protocol. February 2019.
- [3] Gontier Delaunay, M. et al. Morpho Blue Whitepaper. Morpho Labs, October 2023.
- [4] Euler Labs. Euler v2 Lite Paper. 2024.
- [5] Instadapp. Fluid Technical Documentation. 2024. <https://docs.fluid.instadapp.io>
- [6] Fitsialos, J. et al. ERC-4626: Tokenized Vault Standard. Ethereum Improvement Proposals, 2021.
- [7] Weiss, S. EIP-7201: Namespaced Storage Layout. Ethereum Improvement Proposals, 2023.
- [8] Ellis, S., Juels, A., Nazarov, S. Chainlink: A Decentralized Oracle Network. 2017.
- [9] a16z crypto. Halmos: Symbolic Testing for EVM Smart Contracts. <https://github.com/a16z/halmos>
- [10] Feist, J. et al. Slither: A Static Analysis Framework for Smart Contracts. IEEE WETSEB, 2019.
- [11] Frangela, E., Herskind, L. Aave v3 Technical Paper. January 2022.
- [12] Multyr Foundation. Protocol Source Code. <https://github.com/Multyr>