ack3

SECURITY PARTNER

HyroTrader

Hyro Protocol

ENGAGEMENT	SECURITY REVIEW
REVISION	1.1
DELIVERED	2026-10-08
LEAD AUDITOR	Jan Kalivoda

Contents

1. Document Revisions . . . 3
2. Overview . . . 4
2.1. ack3 . . . 4
2.2. Review Methodology . . . 5
2.3. Finding Classification . . . 6
2.4. Review Team . . . 8
2.5. Disclaimer . . . 8
3. Executive Summary . . . 9
Revision 1.0 . . . 9
Revision 1.1 . . . 10
4. Findings Summary . . . 12
Report Revision 1.0 . . . 22
Revision Team . . . 22
System Overview . . . 22
Trust Model . . . 22
Findings . . . 23
Report Revision 1.1 . . . 208
Revision Team . . . 208
System Overview . . . 208
Trust Model . . . 208
Appendix A: How to cite . . . 210
Appendix B: Review Scope . . . 211

1. Document Revisions

1.0-draft	Draft Report	September 11, 2026
1.0	Final Report	October 6, 2026
1.1	Fix Review	October 8, 2026

2. Overview

This document presents the findings of our review.

2.1. ack3

ack3 is an in-house team of security researchers performing audits focused on manual code reviews with extensive fuzz testing and AI-assisted workflows for Ethereum and Solana. The ack3 team is trusted by top-tier organizations in web3, including Lido, Safe, and Axelar.

We develop the open-source security and developer tooling Wake, which powers the fuzzing, static analysis, and graph-based review on our audits and helps auditors discover hardly recognizable edge-case vulnerabilities.

Our team teaches blockchain security at the Czech Technical University in Prague, led by founder Josef Gattermayer, Ph.D.

For ack3, security is a long-term partnership: we work alongside our clients from design through post-deployment.

2.2. Review Methodology

We begin each review by mapping the codebase and the interactions between contracts and external components using our own tooling. We work with the provided documentation and materials to discover discrepancies between the expected behavior and the actual implementation. Where applicable, we build a mathematical model of the protocol to cross-compare with the code and identify subtle bugs and inconsistencies.

At the start of the review, we launch an automated ack3 AI scan pinned to the scoped commits, running in parallel with the manual review. The scan provides additional coverage and new leads for the team to analyze; each scan finding is manually reviewed by the team.

During the review, we apply AI-assisted workflows and skills that support codebase onboarding and navigation.

We launch AI agents to build on our vulnerability hypotheses and expand the attack surface with new escalation paths. All findings are manually reviewed and validated by the team. Where applicable, we confirm findings and verify the exact behavior of the code with unit tests and fuzzing. We use the wake.sol fuzzer for protocol testing. Where the engagement includes a fuzzing campaign, it is manually guided by an auditor; the fuzz tests themselves are AI-generated.

Once sufficient context is gathered, we search for deeper, chained issues based on the attack vectors discovered during review. This includes combining smaller weaknesses, testing trust-boundary assumptions, and evaluating whether unexpected behavior can lead to practical impact.

The validated results are documented in the report: findings with severity, affected components, impact, and recommended remediation.

2.3. Finding Classification

The Severity of each finding combines two sub-ratings, Impact and Likelihood, and ranges from Info to Critical.

If we find a scenario in which an issue is exploitable, it is assigned an impact rating of High, Medium, or Low, based on its consequences for the system. If we find no way to exploit it, or the issue is only exploitable given a change in configuration (system settings or parameters, such as deployment scripts, compiler configurations, or multi-signature wallets for owners) or in the codebase, it is assigned Warning or Info.

Low to High impact issues also have a Likelihood, which measures the probability of exploitation at runtime.

Severity

		LIKELIHOOD			
		HIGH	**MEDIUM**	**LOW**	**N/A**
IMPACT	**HIGH**	Critical	High	Medium	-
	MEDIUM	High	Medium	Low	-
	LOW	Medium	Low	Low	-
	WARNING	-	-	-	Warning
	INFO	-	-	-	Info

Table 1. Severity of findings

Impact

- **High** - Code that activates the issue leads to undefined or catastrophic consequences for the system.
- **Medium** - Code that activates the issue results in serious consequences.
- **Low** - Code that activates the issue has consequences that are recoverable or do not jeopardize the system's normal operation.
- **Warning** - The issue cannot be exploited given the current code or `configuration`, but could become a security vulnerability if either changed. If we have not found a way to exploit the issue given the time constraints, it may be marked as `Warning` or higher, based on our best estimate of whether it is currently exploitable.
- **Info** - The issue is on the borderline between code quality and security. Examples include insufficient logging for critical operations, or issues that would become security-related if code or `configuration` changed.

Likelihood

- **High** - The issue is exploitable by virtually anyone under any circumstance.
- **Medium** - Exploiting the issue requires non-trivial preconditions.
- **Low** - Exploiting the issue requires strict preconditions.

2.4. Review Team

For a revision's authors, see the "Revision team" section in that revision's chapter.

NAME	POSITION
Jan Kalivoda	Lead Auditor
Naoki Yoshida	Auditor
Michal Převrátil	Auditor
Josef Gattermayer, Ph.D.	Audit Supervisor

2.5. Disclaimer

We have made our best effort to find all vulnerabilities in the system; however, our findings should not be considered a complete list of issues. Statements in this document should not be interpreted as investment or legal advice, nor should the authors be held accountable for decisions based on them.

3. Executive Summary

HyroTrader is a Solana protocol for capital deployed to off-chain venues. Depositors subscribe to a vault and receive its shares.

Revision 1.0

HyroTrader engaged ack3 to perform a security review of HyroTrader Hyro Protocol with a total effort of 11 engineering days between August 25 and September 11, 2026, with Jan Kalivoda as the lead auditor.

The review addressed commit `f719260`[1] in the protocol repository. The precise scope is listed in the Review Scope appendix.

We began our review by examining the system architecture and on-chain program logic. In parallel, we prepared manually guided fuzz tests and issue-reproduction proofs of concept using the wake.sol testing framework. We later launched an automated ack3 AI scan.

During the review, we paid special attention to:

- authorization and account constraints for deposits, withdrawals, fund movements, fee claims, and vault-signed transactions;
- permissionless and policy-controlled operations that could enable griefing, unauthorized state transitions, or unauthorized capital movement;
- consistency among token custody, capital in transit, reported off-chain equity, accrued fees, NAV, and share supply;
- oracle, administrator, and manager controls affecting share pricing, settlement, emergency operations, and authority handovers; and
- account and data validation, PDA collisions, arithmetic boundaries, state-machine errors, and unsafe migrations.

Our review resulted in 80 findings, ranging from Info to Critical severity. The most severe findings were C1, which enables an unauthorized vault drain, and C2, which allows repeated report fees to consume all shareholder equity. The findings clustered in authorization, data validation, state management, and fee accounting, and repeatedly identified implemented behavior that diverged from the documentation.

ack3 recommends HyroTrader:

- create and maintain a detailed protocol specification, and align all inline documentation with the implementation;
- expand authorization, fee-accounting, NAV-accounting, state-transition, and migration tests, and run them in CI;
- read and review the complete report; and
- address all identified issues.

See Report Revision 1.0 for the system overview and trust model.

Revision 1.1

HyroTrader engaged ack3 to perform a fix review of the findings from the previous revision.

HyroTrader provided fixes and comments to the acknowledged findings.

The review was performed on October 6, 2026, at commit `a21df9a`[2].

Of the 80 reported findings, 36 were fully fixed, 23 were acknowledged, and 21 remained reported because they were either unresolved or only partially fixed.

See Report Revision 1.1 for more details.

[1] full commit hash: `f7192606295304d65ed2c727a5cb6e35bd83ed34`

[2] full commit hash: `a21df9ab3430b3d69adaa75e5f8b3fa13cb1d272`

4. Findings Summary

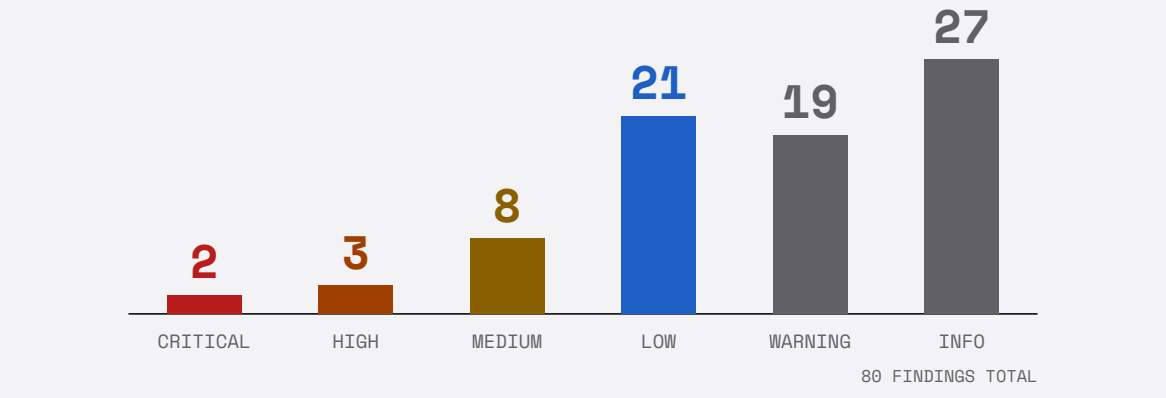

Figure 1. Findings by Severity

FINDING TITLE	SEVERITY	REVISION	STATUS
C1: Missing authorization and ineffective limits allow a full vault drain	CRITICAL	1.0	Acknowledged
C2: Fees on balance reports erode vault equity	CRITICAL	1.0	Reported
H1: Repeated internal transfers drain vault value through fees	HIGH	1.0	Fixed
H2: Former administrators can drain vault assets through fees	HIGH	1.0	Fixed
H3: Misallocated fees let transactors drain other shareholders' value	HIGH	1.0	Fixed

FINDING TITLE	SEVERITY	REVISION	STATUS
M1: Unpayable report fees understate challenge equity	MEDIUM	1.0	Acknowledged
M2: Absolute reports erase in-transit capital	MEDIUM	1.0	Reported
M3: Unauthorized request decisions can block deposits and withdrawals	MEDIUM	1.0	Acknowledged
M4: Registry takeover blocks manager onboarding and child issuance	MEDIUM	1.0	Fixed
M5: Address collision can permanently block manager registration	MEDIUM	1.0	Fixed
M6: Unauthorized policy setup can block vault operations	MEDIUM	1.0	Acknowledged
M7: Unrelated admin signatures bypass the withdrawal gate	MEDIUM	1.0	Acknowledged
M8: Unbounded oracle reports enable immediate share-price manipulation	MEDIUM	1.0	Acknowledged

FINDING TITLE	SEVERITY	REVISION	STATUS
L1: Unexecutable proposals have no rent-recovery path	LOW	1.0	Fixed
L2: Challenge administrators can appoint child equity oracles	LOW	1.0	Fixed
L3: Unbound challenge registration blocks policy setup	LOW	1.0	Fixed
L4: Child and transaction addresses collide	LOW	1.0	Fixed
L5: Proposal execution forwards the executor's signature	LOW	1.0	Fixed
L6: Invalid fee program configuration blocks fund operations	LOW	1.0	Reported
L7: Invalid calculator configuration halts fee-charging operations	LOW	1.0	Reported
L8: Unchecked policy selection interrupts vault operations	LOW	1.0	Acknowledged

FINDING TITLE	SEVERITY	REVISION	STATUS
L9: Delayed approvals authorize recycled proposals	LOW	1.0	Acknowledged
L10: One multisig owner can veto threshold-approved proposals	LOW	1.0	Acknowledged
L11: Direct returns can be omitted from NAV if the oracle reports first	LOW	1.0	Fixed
L12: Request replacement bypasses rejection	LOW	1.0	Acknowledged
L13: Retired administrators retain manager access	LOW	1.0	Acknowledged
L14: Retired and active policy binaries use the same address	LOW	1.0	Acknowledged
L15: Unnecessary writable accounts couple independent vaults	LOW	1.0	Reported
L16: Transfers above the signed limit can reverse balance updates	LOW	1.0	Fixed
L17: One multisig owner can disable withdrawals	LOW	1.0	Acknowledged

FINDING TITLE	SEVERITY	REVISION	STATUS
L18: Single-step handovers can lose administrative access	LOW	1.0	Fixed
L19: Exchange debt cannot be reflected in NAV	LOW	1.0	Reported
L20: Fixed virtual assets distort final redemptions	LOW	1.0	Reported
L21: Unset recipients can lock accrued fees	LOW	1.0	Fixed
W1: Vault administrator can replace multisig membership alone	WARNING	1.0	Acknowledged
W2: Unauthenticated calls corrupt calculator fee state	WARNING	1.0	Fixed
W3: Closed challenge identities can be reused	WARNING	1.0	Reported
W4: Child allocation can leave the parent insolvent	WARNING	1.0	Fixed
W5: CI omits account-layout regression checks	WARNING	1.0	Fixed
W6: Fee claims inherit capital-return permissions	WARNING	1.0	Fixed

FINDING TITLE	SEVERITY	REVISION	STATUS
W7: Fee dispatch receives custody authority	WARNING	1.0	Fixed
W8: Timestamp resets discard unsettled fee periods	WARNING	1.0	Reported
W9: Performance calculations misclassify principal as profit	WARNING	1.0	Reported
W10: Child deposits can open without a usable share mint	WARNING	1.0	Reported
W11: Manager permissions grant full staged-fund custody	WARNING	1.0	Acknowledged
W12: Missing signature context blocks approval-gated vault operations	WARNING	1.0	Fixed
W13: Policy helpers leave ownership checks to callers	WARNING	1.0	Reported
W14: Authorized proposals have no enforced response window	WARNING	1.0	Acknowledged
W15: Proposal approval does not bind target program code	WARNING	1.0	Acknowledged

FINDING TITLE	SEVERITY	REVISION	STATUS
W16: Request handlers omit core policy-state binding	WARNING	1.0	Fixed
W17: Policy replacement preserves pending proposals	WARNING	1.0	Acknowledged
W18: Time-fee calculations misprice unsettled periods	WARNING	1.0	Acknowledged
W19: Account growth can preserve stale field values	WARNING	1.0	Acknowledged
I1: Reporting equity before bootstrap blocks the first deposit	INFO	1.0	Reported
I2: Parent reports fail with an unrelated account-type error	INFO	1.0	Acknowledged
I3: Documented challenge seeds exceed the PDA limit	INFO	1.0	Fixed
I4: Checkout documentation overstates voucher binding	INFO	1.0	Reported
I5: Core account growth lacks explicit layout safeguards	INFO	1.0	Fixed

FINDING TITLE	SEVERITY	REVISION	STATUS
I6: Dead modules misrepresent supported core features	INFO	1.0	Fixed
I7: Deny-all policy still permits transaction proposals	INFO	1.0	Fixed
I8: Duplicated core IDs can desynchronize authorization checks	INFO	1.0	Acknowledged
I9: Escrow bump comments contradict account declarations	INFO	1.0	Fixed
I10: Escrow rent and surplus balances cannot be reclaimed	INFO	1.0	Acknowledged
I11: Time and performance fees have no core accrual path	INFO	1.0	Reported
I12: Fee totals use inconsistent boundary arithmetic	INFO	1.0	Fixed
I13: Accrual fee cap permits the entire vault NAV	INFO	1.0	Reported

FINDING TITLE	SEVERITY	REVISION	STATUS
I14: Manager registry fields do not track ongoing activity	INFO	1.0	Reported
I15: Manager verification returns an inconsistent authorization error	INFO	1.0	Fixed
I16: Validation errors misidentify the failed constraint	INFO	1.0	Fixed
I17: Overlapping error codes misidentify rejected operations	INFO	1.0	Fixed
I18: Amount policies must decode core instructions	INFO	1.0	Reported
I19: Policy validation does not authenticate core provenance	INFO	1.0	Reported
I20: Delay changes extend existing withdrawal locks	INFO	1.0	Reported
I21: Build caching omits shared Rust crates	INFO	1.0	Fixed
I22: Signature lookup rejects valid transaction layouts	INFO	1.0	Fixed

FINDING TITLE	SEVERITY	REVISION	STATUS
I23: SDK always enables IDL-generation features	INFO	1.0	Fixed
I24: Unpinned token accounts produce misleading balance errors	INFO	1.0	Reported
I25: Child policy validation receives unwritten vault state	INFO	1.0	Fixed
I26: Vault lifecycle documentation and state are inconsistent	INFO	1.0	Fixed
I27: Vault state omits its virtual balance mode	INFO	1.0	Fixed

Table 2. Table of Findings

Report Revision 1.0

Revision Team

NAME	POSITION
Jan Kalivoda	Lead Auditor
Naoki Yoshida	Auditor
Michal Převrátil	Auditor
Josef Gattermayer, Ph.D.	Audit Supervisor

System Overview

Hyro Protocol is a Solana vault protocol for capital deployed to off-chain trading venues. Depositors receive shares priced against net asset value, which combines recorded on-chain tokens and reported off-chain equity, less accrued fees. Capital moves through `use_funds` and `return_funds`; deposit and withdrawal requests settle asynchronously. The core combines explicit administrator and oracle checks with operation-specific policy calls. A separate fee dispatcher and calculator determine fee charges. Child vaults include token-funded vaults and virtual challenge vaults that record equity without receiving an allocation of tokens.

Trust Model

Users should treat the current vault administrator, fee authority, designated oracle, and every manager authorized by the active policy as trusted roles. The administrator can replace the policy and oracle and change how the vault operates. The fee authority can set a combined fee rate of up to 100%, choose who receives those fees, and may remain controlled by a former

administrator after a handover (H2). The oracle supplies the off-chain equity used to price every share, so a false report can distort deposits and withdrawals (M8). Under the manager-access policy, appointed managers can transfer staged capital to destinations they choose (W11). The selected policy determines who can move capital or execute vault-signed transactions; an unsafe policy can expose funds to arbitrary callers (C1). These roles can freeze withdrawals, misprice shares, or transfer and collect user value. Before depositing, users should verify and trust the active keys, policy, fee configuration, and handover practices. On-chain configuration cannot prove that reported assets exist at the off-chain trading venue.

Findings

This section lists the findings from this revision; for all findings, see the Findings Summary.

C1: Missing authorization and ineffective limits allow a full vault drain

CRITICAL SEVERITY ISSUE

Impact:	High	**Likelihood:**	High
Type:	Data validation	**Status:**	Acknowledged
Target:	programs/policy_limit_transfer/src/lib.rs		

Description

`policy_limit_transfer` neither authenticates callers nor correctly enforces its configured `[min, max]` transfer band, although `hyro_protocol` delegates authorization for `use_funds`, `create_tx`, and `execute_tx` to `Vault::policy_program`.

Seven `validate` arms check only the account-list length and return `Ok(())`: `UseFunds`, `ReturnFunds`, `Report`, `RequestWithdrawal`, `ProcessWithdrawals`, `RequestDeposit`, and `ProcessDeposits`.

Listing 1. Excerpt from The UseFunds arm delegates to the length-only account check

```
65 ValidateContext::UseFunds(use_funds_ctx) => {
66     use_funds_ctx.validate_accounts(ctx.remaining_accounts)?;
67     Ok(())
68 }
69 ValidateContext::ReturnFunds(return_funds_ctx) => {
70     return_funds_ctx.validate_accounts(ctx.remaining_accounts)?;
71     Ok(())
72 }
```

`validate_accounts` compares `remaining_accounts.len()` with an index but never reads or authenticates the signer slot.

Listing 2. Excerpt from The shared account check validates only the account-list length

```
389 pub fn validate_accounts(&self, remaining_accounts: &[AccountInfo]) ->
    Result<()> {
390     require!(
391         remaining_accounts.len() > self.policy_account_index,
392         ErrorCode::MissingPolicyAccount
393     );
394     require!(
395         remaining_accounts.len() > self.signer_index,
396         ErrorCode::MissingSigner
397     );
398     Ok(())
399 }
```

`Creation` additionally validates the policy-state PDA but does not authenticate the caller or inspect the proposed instruction. Only `IssueChildVault` fails closed.

`Execution` checks the policy-state PDA and transaction owner, but decodes the transfer amount from `payload[8..16]`.

Listing 3. Excerpt from The Execution arm reads bytes 8 through 15 as the transfer amount

```
49 let amount = payload
50     .get(8..16)
51     .and_then(|s| s.try_into().ok())
52     .map(u64::from_le_bytes)
53     .ok_or(error!(ErrorCode::InvalidData))?;
54
55 if amount > max {
56     return Err(ErrorCode::AmountTooHigh.into());
57 }
58
59 if amount < min {
60     return Err(ErrorCode::AmountTooLow.into());
61 }
```

An SPL Token `Transfer` payload contains a one-byte tag followed by a little-endian `u64` amount in bytes 1 through 8. SPL Token 7.0.0, pinned in

`apps/protocol/Cargo.lock:2553-2557`, decodes the first eight bytes after the tag and ignores trailing bytes. The policy instead interprets byte 8, the transfer amount's most significant byte, followed by seven attacker-controlled trailing bytes.

For amounts below `2^56`, byte 8 is zero. An attacker can therefore append bytes that make the policy read an unrelated in-band multiple of 256 while SPL Token reads the intended transfer amount. A canonical nine-byte payload cannot satisfy `payload.get(8..16)` and is rejected with `InvalidData`, so the configured band never bounds a well-formed transfer.

Exploitation requires successful `use_funds` fee charging and bookkeeping, enabled proposed transactions, and an initialized band that admits the shared-byte encoding. The React SDK creates vaults with `LimitTransferPolicy({min: 0.1, max: 1000000000})` (`packages/react-sdk/src/hooks/use-create-new-vault.ts:6-23`). An administrator can also select this executable policy through `update_vault_policy` after disabling transactions; the setter does not enforce a policy allowlist (`programs/hyro_protocol/src/vault.rs:1139-1152`, `programs/hyro_protocol/src/vault.rs:1287-1310`). An impossible or sufficiently narrow band does not admit the encoding.

The resulting instruction can transfer the full balance of `authority_token_account`: `execute_tx` marks each account matching `Vault::authority` as a signer and invokes the instruction with that PDA's signature. This route does not directly access `vault_token_account`, whose token authority is `share_signer_pda`.

Exploit scenario

1. Bob, the vault administrator, configures `policy_limit_transfer`, enables proposed transactions, and holds 5,000,000 base units of working capital in `authority_token_account`.

2. Alice, an unprivileged account holder with enough funds for fees and proposal rent, invokes `use_funds` to move available working capital from `vault_token_account` to `authority_token_account` if Bob has not already done so. The length-only `UseFunds` validation accepts her call when fee charging and balance bookkeeping succeed.
3. Alice invokes `create_tx` with a 16-byte SPL Token `Transfer` from `authority_token_account` to her token account. Bytes 1 through 8 encode 5,000,000, while bytes 8 through 15 encode 1,499,904, which lies within a configured band of 1,000,000 to 2,000,000. `Creation` accepts the proposal without authenticating Alice or inspecting its payload.
4. Alice invokes `execute_tx`. `Execution` approves 1,499,904, `execute_tx` signs for `Vault::authority`, and SPL Token transfers the 5,000,000 base units encoded at bytes 1 through 8 to Alice.

Recommendation

1. Authenticate the caller in every policy arm that grants authorization. In `UseFunds`, `ReturnFunds`, `Report`, `Creation`, and `Execution`, use the SDK `signer` accessor to assert `is_signer` and require an administrator or registered manager.
2. Enforce the limit in `hyro_protocol::execute_tx` by comparing the pre-execution and post-execution balance of `authority_token_account`. The policy CPI runs before the proposed instruction and has no post-execution hook, whereas `execute_tx` can observe both states in one call frame.
3. Require `Vault::admin` or a registered manager to sign `use_funds` and `return_funds` within `hyro_protocol`, independently of the policy verdict.
4. Use an authenticated policy as the vault-provisioning default. Identify vaults that use `policy_limit_transfer`, inspect their configurations, and migrate affected vaults through the administrator-controlled pause and policy-replacement instructions.

Acknowledgment 1.1

> `policy_limit_transfer` is test contract out of audition scope.
>
> Not deployed to real network
>
> — HyroTrader Team

Go back to Findings Summary

C2: Fees on balance reports erode vault equity

CRITICAL SEVERITY ISSUE

Impact:	High	**Likelihood:**	High
Type:	Logic error	**Status:**	Reported
Target:	programs/hyro_protocol/src/funds.rs		

Description

`report_on_funds` treats the absolute change in `Vault::offchain_balance` as a subscription fee basis even though a report only records valuation and transfers no tokens.

Listing 4. Excerpt from Absolute reported change used as the fee basis

```
1697 let fee_amount = match balance {
1698     BalanceUpdate::Delta(delta) => delta.unsigned_abs(),
1699     BalanceUpdate::NewValue(value) =>
     value.abs_diff(ctx.accounts.vault.offchain_balance),
1700 };
```

The handler submits that basis to the fee subsystem as a subscription and updates `Vault::obtained_fees` before installing the reported balance.

Listing 5. Excerpt from Balance report charged as a subscription

```
1709     FeeOperation::Deposit,
1710     fee_amount,
1711     ctx.accounts.vault.total_balance,
1712     ctx.accounts.vault.high_water_mark,
1713     ctx.accounts.vault.last_fee_collection as i64,
1714 )?;
1715 ctx.accounts.vault.update_obtained_fees(obtained)?;
```

Because `unsigned_abs` and `abs_diff` discard direction, a decrease incurs the same rate as an increase, and an oscillation that returns equity to its starting

value incurs a fee on each leg despite producing no net appreciation.

The fractional strategy applies its rate unconditionally because `FractionFeeConfig` has no per-operation flag.

The reporting component supplies the venue's absolute `totalEquityUsd`, converted to underlying-token raw units, through a `NewValue` update rather than reporting a transfer or performance delta.

Listing 6. Excerpt from Reporting component supplies an absolute balance

```
73 async reportOnFunds(offchainBalance: bigint): Promise<string> {
74   const ixs = await this.vault.buildReportOnFundsInstruction({
75     balance: { __kind: 'NewValue', fields: [offchainBalance] },
76   });
```

`Vault::obtained_fees` reduces net asset value and has no cumulative cap based on available tokens.

Repeated reports can therefore consume all remaining equity, produce negative net asset value, and block settlement; configured fee recipients can claim backing tokens while sufficient liquidity remains.

The fee-first write order also rejects some valid final states because each setter immediately recalculates signed net asset value against the current fields.

From an off-chain balance of one, a `NewValue(u64::MAX)` report at a 100% rate has final net asset value of one, but applying the fee against the old balance first falls below `i64::MIN` and reverts; smaller reports can reach the same final state.

A report requires the signer to equal `vault_oracle.oracle`, the oracle account to identify the vault or its parent, and the policy's `Report` arm to approve the operation (`programs/hyro_protocol/src/funds.rs:1684-1693`).

`policy_checkout` routes `Report` through its administrator gate, so its reporter must also be the vault administrator, whereas `policy_limit_transfer` requires no additional policy role (`programs/policy_limit_transfer/src/lib.rs:79-82`).

These checks restrict who can choose reports but do not prevent routine honest valuation changes from accruing the fees.

Exploit scenario

1. Bob, the designated oracle who satisfies the applicable `Report` policy, follows the normal reporting process as ordinary trading changes a vault's off-chain equity from zero to 1,000,000 units and later back to zero.
2. Bob submits the corresponding `NewValue(1_000_000)` and `NewValue(0)` reports to a vault with no on-chain balance and a 50-basis-point combined rate.
3. Each honest report accrues 5,000 units, leaving `Vault::offchain_balance` at zero, `Vault::obtained_fees` at 10,000, and net asset value at -10,000.
4. Alice, a vault holder, bears the resulting liability through reduced net asset value, and settlement remains blocked while the deficit persists. No adversary or abnormal transaction sequence is required; ordinary market movement and correct oracle reporting are sufficient.

Recommendation

1. Remove subscription-rate fee accrual from `report_on_funds`, and keep balance-report valuation changes outside subscription fee accounting.
2. If valuation-linked fees are required, charge them only on net appreciation above a high-water mark and exclude reported declines.
3. Enforce the invariant recommended in H1 in `Vault::update_obtained_fees` to cap this accrual path.
4. Compute and range-check final net asset value from the proposed balance and fee liability before committing either field.

Update 1.1

Report fees are disabled by default, but an enabled report fee still charges each loss recovery instead of net appreciation and can erode equity during a flat-performance oscillation.

Go back to Findings Summary

H1: Repeated internal transfers drain vault value through fees

HIGH SEVERITY ISSUE

Impact:	High	**Likelihood:**	Medium
Type:	Logic error	**Status:**	Fixed
Target:	programs/hyro_protocol/src/funds.rs		

Description

`use_funds` and `return_funds` move SPL tokens between vault-controlled accounts without changing gross holdings, but each operation charges a percentage fee on the full transferred amount:

Listing 7. Excerpt from Fee charged on the returned amount

```
1512 let obtained = invoke_charge_fees(
1513     &ctx.accounts.vault.to_account_info(),
1514     &ctx.accounts.fee_collection_program.to_account_info(),
1515     &ctx.accounts.share_signer_pda.to_account_info(),
1516     ctx.accounts.vault.key(),
1517     ctx.bumps.share_signer_pda,
1518     fee_accounts,
1519     FeeOperation::UseFunds,
1520     assets,
1521     ctx.accounts.vault.total_balance,
1522     ctx.accounts.vault.high_water_mark,
1523     ctx.accounts.vault.last_fee_collection as i64,
1524 )?;
1525 ctx.accounts.vault.update_obtained_fees(obtained)?;
1526 ctx.accounts.vault.update_last_fee_collection()?;
```

The fee accrues in `Vault::obtained_fees` without transferring tokens to a recipient. `recalculate_total_balance` subtracts this liability from the vault's on-chain and off-chain balances to calculate NAV:

Listing 8. Excerpt from Accrued fees deducted from gross holdings

```
248 fn recalculate_total_balance(&mut self) -> Result<()> {
249     let gross = (self.onchain_balance as i128) + (self.offchain_balance as
    i128);
250     let net = gross - (self.obtained_fees as i128);
251     self.total_balance = i64::try_from(net).map_err(|_|
    error!(ErrorCode::MathOverflow))?;
252     Ok(())
```

The fractional fee strategy applies the same mandatory rate to `Deposit`, `Withdraw`, `UseFunds`, and `ReturnFunds`; `FractionFeeConfig` has no per-operation flags. The per-call ceiling in `charge_fees_cpi` only requires `output.fees.total() ⇐ input.amount` and does not impose a cooldown or cumulative ceiling. Because each round trip restores the principal for reuse, repeated calls accrue twice the fee rate per round trip while gross holdings remain unchanged.

The program permits `obtained_fees` to exceed gross holdings. NAV then becomes negative, preventing subscriptions and redemptions from settling. Claiming the fees reduces gross holdings and the liability equally, so it does not restore NAV. Restoring settlement requires new capital sufficient to cover the deficit and any fee charged on its return.

Exploit scenario

1. Bob, the current vault administrator, configures a 50-basis-point fractional fee.
2. Alice, an authorized operator, repeatedly moves the full on-chain balance with `use_funds` and returns it with `return_funds`. Each round trip preserves gross holdings but adds fees equal to 1% of the reused principal. If the permissive `policy_limit_transfer` arms from C1 apply, any signer can trigger this churn.
3. Shareholder NAV reaches zero after approximately 100 round trips.

4. Further accrual makes NAV negative, so Carol, a shareholder, cannot redeem and new subscriptions do not settle. Only a configured fee recipient can claim the proceeds.

Recommendation

1. Exclude `FeeOperation::UseFunds` and `FeeOperation::ReturnFunds` from percentage fees, or add corresponding per-operation flags to `FractionFeeConfig`.
2. If movement fees are required, enforce a minimum interval and per-epoch cumulative ceiling in `charge_fees_cpi`, using `fees.last_update`.
3. Enforce `obtained_fees ⇐ onchain_balance + offchain_balance` in `Vault::update_obtained_fees` so no fee-accrual path can produce negative NAV.

Fix 1.1

The fraction calculator now makes internal-movement and report fees optional and disables them by default while retaining deposit and withdrawal fees.

Go back to Findings Summary

H2: Former administrators can drain vault assets through fees

HIGH SEVERITY ISSUE

Impact:	High	**Likelihood:**	Medium
Type:	Logic error	**Status:**	Fixed
Target:	programs/hyro_protocol/src/vault.rs, programs/fee_collection/src/lib.rs		

Description

`set_vault_admin` updates `Vault::admin` without rotating the separate authorities copied into the fee ledger and active calculator during initialization or legacy migration.

Listing 9. Excerpt from Fee ledger copies the current vault administrator

```
252 fees.authority = vault.admin;
253 Ok(())
```

Listing 10. Excerpt from Vault handover does not update fee authority

```
1278 pub fn set_vault_admin(ctx: Context<UpdateVaultStatus>, new_admin:
     Pubkey) -> Result<()> {
1279     require!(new_admin != Pubkey::default(), ErrorCode::InvalidAdmin);
1280     ctx.accounts.vault.admin = new_admin;
1281     Ok(())
1282 }
```

Fee mutators authenticate these stored authorities instead of the live `Vault::admin`:

Listing 11. Excerpt from Fee changes use the retained authority

```
155 #[account(
```

```
156     mut,
157     seeds = [b"vault_fees", vault.key().as_ref()],
158     bump = vault_fees.bump,
159     constraint = vault_fees.authority == authority.key() @
    ErrorCode::Unauthorized,
160 )]
161 pub vault_fees: Account<'info, VaultFees>,
162
163 pub authority: Signer<'info>,
```

An outgoing administrator therefore retains control of fee recipients, the dispatcher, and the active calculator after the handover, while the incoming administrator cannot revoke that control without the outgoing administrator's signature through the separate `update_authority` instructions. Legacy migrations create the same split when they precede an administrator rotation; they copy the administrator once and return early for current-layout accounts, so rerunning them cannot reconcile the authority, and the migration caller cannot nominate another key.

Subsequent fee-bearing operations accrue liability under the former administrator's configuration, reduce shareholder NAV, and allow `claim_fees` to transfer available backing tokens to the configured recipients. Ordinary activity by the new administrator can fund those recipients. The permissive policy in C1 lets any signer trigger internal movements and fee claims, while the fee basis in H1 lets repeated `use_funds` and `return_funds` calls reuse the same principal, amplifying the loss.

Exploit scenario

1. Alice, the current administrator, initializes the fee ledger and calculator and then transfers `Vault::admin` to Bob without rotating their authorities.
2. Alice remains authorized to direct fee recipients to herself, raise the fractional rate, and select the dispatcher and active calculator.
3. Bob performs ordinary fee-bearing operations, which accrue liabilities under Alice's configuration and reduce shareholder NAV.

4. Alice claims the available backing tokens; under a permissive policy, she or any signer can accelerate accrual through repeated internal transfers.
5. Bob cannot assume or revoke the retained authorities without Alice's cooperation.

Recommendation

1. Rotate the vault administrator and every fee authority atomically, including the active calculator, or authenticate fee mutations against the live `Vault::admin`.
2. Add a recovery path that lets the current administrator replace obsolete authorities in existing ledgers and calculators without the outgoing key.
3. Make migrations reconcile fee authorities deliberately and idempotently regardless of their timing relative to an administrator handover.
4. Apply the independent authorization and fee-basis fixes in C1 and H1.

Fix 1.1

Fee calculator configuration changes now authenticate the live, owner-checked `Vault.admin`, so a former stored authority cannot retain control after a vault administrator change.

Go back to Findings Summary

H3: Misallocated fees let transactors drain other shareholders' value

HIGH SEVERITY ISSUE

Impact:	High	**Likelihood:**	Medium
Type:	Logic error	**Status:**	Fixed
Target:	programs/hyro_protocol/src/funds.rs		

Description

Both settlement handlers calculate the transacting party's shares or payout before adding the transaction fee to `Vault::obtained_fees`.

`Vault::obtained_fees` is an accrued liability rather than a token transfer, so recording it reduces NAV for the outstanding shares.

The subscription handler mints shares against the full deposit and records the entry-fee liability only after crediting the deposited assets:

Listing 12. Excerpt from Subscription accrues the fee after minting shares

```
1038 token::mint_to(mint_ctx, shares)?;
1039
1040 ctx.accounts
1041     .vault
1042     .update_onchain_balance(BalanceUpdate::Delta(assets as i64))?;
1043
1044 // Now that the deposit is on the books, apply the fee liability computed
     above.
1045 ctx.accounts.vault.update_obtained_fees(obtained)?;
1046 ctx.accounts.vault.update_last_fee_collection()?;
```

The subscriber therefore bears only the portion of the entry fee attributable to the newly minted shares, while existing shareholders bear the remainder.

The redemption handler calculates `assets` before recording the exit fee, then

burns the shares and transfers the unchanged pre-fee amount:

Listing 13. Excerpt from Redemption transfers the pre-fee asset amount

```
1316     share_signer_seeds,
1317 );
1318 token::transfer(transfer_ctx, assets)?;
1319
1320 ctx.accounts
1321     .vault
1322     .update_onchain_balance(BalanceUpdate::Delta(-(assets as i64)))?;
```

The redeemer bears none of the exit fee because the corresponding shares are burned after the liability is recorded, leaving retained shareholders to bear it.

A near-total redemption can also pass the raw-token balance check but leave negative NAV after the token debit and new fee liability.

Deposits reject negative NAV, and redemptions require strictly positive NAV, so this state can block subsequent settlements (`programs/hyro_protocol/src/funds.rs:910-925`, `programs/hyro_protocol/src/funds.rs:1226-1267`).

Subscription must continue using pre-deposit NAV because booking the liability before crediting the deposit would make a fresh vault's NAV negative and cause `convert_to_shares` to reject it (`programs/hyro_protocol/src/vault.rs:274-277`).

Exploit scenario

1. Bob, the vault operator and fee recipient, configures a combined fee rate of 50 basis points.
2. Alice, a user permitted to settle both legs with no withdrawal delay, deposits 100,000 units into a vault with 1,000,000 units and 1,000,000 shares. Ignoring virtual offsets and rounding, the 500.00 entry fee is

recorded after Alice receives 9.09% of the shares, so Alice bears 45.45 while retained shareholders bear 454.55.

3. Alice immediately redeems her new shares. The redemption is priced at 99,954.55, and the handler transfers that full amount despite accruing a 499.77 exit fee.
4. The two fees total 999.77, but Alice loses only 45.45; retained shareholders lose 954.32 of NAV, which accrues to Bob as fees.
5. Alice repeats smaller cycles using the returned principal, reducing her proportional entry-fee cost while repeatedly shifting the exit fee and most of each entry fee to retained shareholders.
6. A near-total redemption independently makes NAV negative. With 1,000,000,000 raw units initially deposited, equal share supply, and a 1% entry fee, NAV is 990,000,000.
7. Alice redeems all shares, which pays 990,009,990 under the virtual-offset formula and adds a 9,900,099 exit-fee liability, leaving NAV at -9,910,089 even though the gross-token payout check passes.
8. Further deposits and redemptions fail until sufficient capital or an authorized accounting correction restores the required NAV state; paying accrued fees is NAV-neutral.

Recommendation

1. Charge each transaction's fee to the transacting party by minting subscription shares against `assets - fee` and paying redeemers `assets - fee`.
2. Continue pricing subscriptions against the pre-deposit balance and record the liability only after crediting the deposit.
3. If fees are intentionally distributed across shareholders, document that incidence and cap accrual so settlement cannot create negative NAV.
4. Check solvency after the complete settlement, including the token debit

and new fee liability, and define a reconciliation procedure that requires exactly zero NAV before reopening a zero-supply vault without granting the next depositor a claim on residual assets.

Fix 1.1

Deposit shares and withdrawal payouts now use amounts net of fees, settlement checks final solvency, and a full redemption uses the vault's real net asset value.

Go back to Findings Summary

M1: Unpayable report fees understate challenge equity

MEDIUM SEVERITY ISSUE

Impact:	Medium	**Likelihood:**	Medium
Type:	Logic error	**Status:**	Acknowledged
Target:	programs/hyro_protocol/src/funds.rs, programs/policy_challenges/src/policy.rs		

Description

`report_on_funds` assesses a `Deposit` fee on each change in reported off-chain equity, including both increases and decreases:

Listing 14. Excerpt from Reports charge deposit fees on the equity change

```
1697 let fee_amount = match balance {
1698     BalanceUpdate::Delta(delta) => delta.unsigned_abs(),
1699     BalanceUpdate::NewValue(value) =>
     value.abs_diff(ctx.accounts.vault.offchain_balance),
1700 };
1701
1702 let obtained = invoke_charge_fees(
1703     &ctx.accounts.vault.to_account_info(),
1704     &ctx.accounts.fee_collection_program.to_account_info(),
1705     &ctx.accounts.share_signer_pda.to_account_info(),
1706     ctx.accounts.vault.key(),
1707     ctx.bumps.share_signer_pda,
1708     fee_accounts,
1709     FeeOperation::Deposit,
1710     fee_amount,
```

This fee calculation also applies to virtual challenge vaults, which begin with accounting equity but receive no tokens:

Listing 15. Excerpt from Virtual children hold accounting equity only

```
1027 if virtual_balance {
1028     // Demo trading: the vault anchors off-chain equity and holds no
```

```
    tokens.
1029        child_vault.update_onchain_balance(BalanceUpdate::NewValue(0))?;
1030
    child_vault.update_offchain_balance(BalanceUpdate::NewValue(allocation))?
    ;
1031    } else {
1032
    child_vault.update_onchain_balance(BalanceUpdate::NewValue(allocation))?;
1033        child_vault.update_offchain_balance(BalanceUpdate::NewValue(0))?;
1034    }
```

When the operator provisions a child fee ledger and assigns a calculator with a nonzero applicable `Deposit` rate, each sufficiently large positive or negative report change increases the cumulative fee liability.

The vault subtracts this liability from reported equity, so offsetting gains and losses leave `total_balance` below the oracle-reported value.

The virtual child cannot settle the liability through the normal fee path because it has no vault token account, `claim_fees` requires that account, and the challenge policy rejects the claim's `ReturnFunds` operation.

Exploit scenario

1. Bob, an operator, configures a virtual challenge with a nonzero `Deposit` fee and the required child fee ledger.
2. The oracle reports an equity increase followed by an equal decrease.
3. Both changes accrue fees, so the published vault equity falls below its starting value despite a flat trading result, and the child holds no tokens with which to settle the liability.

Recommendation

Exempt virtual reporting-only vaults from fee accrual and permit zero-fee reports without fee accounts.

If virtual challenges must accrue fees, track gross reported equity separately

from fee-adjusted equity and implement a settlement mechanism that does not require the virtual child to hold tokens.

Enforce a zero `Deposit` fee for every virtual challenge until that settlement mechanism exists.

Acknowledgment 1.1

> `Virtual challenges are deployed with zero fees`
>
> `— HyroTrader Team`

Go back to Findings Summary

M2: Absolute reports erase in-transit capital

MEDIUM SEVERITY ISSUE

Impact:	Medium	**Likelihood:**	Medium
Type:	Logic error	**Status:**	Reported
Target:	programs/hyro_protocol/src/funds.rs		

Description

`use_funds` transfers tokens from the vault to its authority account and adds the amount to `offchain_balance`:

Listing 16. Excerpt from Funds in transit are recorded as off-chain balance

```
1542 ctx.accounts
1543     .vault
1544     .update_onchain_balance(BalanceUpdate::Delta(-(assets as i64)))?;
1545 ctx.accounts
1546     .vault
1547     .update_offchain_balance(BalanceUpdate::Delta(assets as i64))?;
```

An absolute oracle report then overwrites the full `offchain_balance`:

Listing 17. Excerpt from Absolute report overwrites off-chain balance

```
1718 match balance {
1719     BalanceUpdate::Delta(delta) => {
1720         ctx.accounts
1721             .vault
1722             .update_offchain_balance(BalanceUpdate::Delta(delta))?;
1723     }
1724     BalanceUpdate::NewValue(value) => {
1725         ctx.accounts
1726             .vault
1727             .update_offchain_balance(BalanceUpdate::NewValue(value))?;
1728     }
```

Publishing a snapshot taken before `use_funds` therefore removes the new in-

transit credit while the tokens remain in the authority account. Reconciliation cannot recover this value because it reads only the primary vault token account. Until a later report or authorized return restores the NAV, deposits settle at an understated price and receive excess shares, diluting existing holders when the value returns.

Exploit scenario

1. Bob, an oracle operator, snapshots the exchange balance.
2. Alice calls `use_funds`, which moves capital to the authority account and increases `offchain_balance`.
3. Bob publishes the older absolute snapshot and erases that increase.
4. Alice, if authorized to deposit, settles at the depressed price and receives excess shares before the capital is reported or returned. The same state transition can result from ordinary operator timing.

Recommendation

Track in-transit capital separately from exchange equity. Bind each absolute report to a movement sequence or report epoch, and reject snapshots that predate an unsettled transfer. Retire the in-transit receivable only when the exchange confirms receipt or the capital is returned.

Update 1.1

The protected `NewValueIfPrev` variant was added, but unconditional `NewValue` remains available and the supplied Bybit reporter still uses it.

Go back to Findings Summary

M3: Unauthorized request decisions can block deposits and withdrawals

MEDIUM SEVERITY ISSUE

Impact:	Low	**Likelihood:**	High
Type:	Access control	**Status:**	Acknowledged
Target:	programs/hyro_protocol/src/funds.rs		

Description

The approval and rejection contexts accept any signer because `owner` only derives the request address and does not sign:

Listing 18. Excerpt from Request owner is unchecked and used only as the PDA seed

```
424 /// CHECK: request owner; used only as the deposit_request PDA seed.
425 pub owner: UncheckedAccount<'info>,
```

The SDK maps approval and rejection to the ordinary request policy contexts, so policies that allow users to create requests can also authorize request decisions:

Listing 19. Excerpt from Approval and rejection reuse request contexts

```
101 ValidateOperation::ApproveDeposit | ValidateOperation::RejectDeposit => {
102     ValidateContext::RequestDeposit(RequestDepositContext {
103         policy_account_index: 0,
104         signer_index: 1,
105         deposit_request_index: 2,
106         instructions_sysvar_index: 3,
107     })
108 }
109 ValidateOperation::ApproveWithdrawal | ValidateOperation::RejectWithdrawal
    => {
110     ValidateContext::RequestWithdrawal(RequestWithdrawalContext {
111         policy_account_index: 0,
```

```
112         signer_index: 1,
113         withdrawal_request_index: 2,
114         instructions_sysvar_index: 3,
115     })
116 }
```

This affects the limit-transfer, allow-any, withdrawal-window, and signature-gate paths; the checkout policy explicitly allows deposit approval and rejection. Owner, manager-access, and multisig policies instead check the raw operation.

An unauthorized signer can reject a pending deposit or withdrawal request and prevent its settlement. The owner can cancel the rejected request to recover escrow but must create and pay for a replacement. Unauthorized approval has no equivalent effect because both `Pending` and `Approved` requests can settle, and only pending requests can be approved or rejected.

Exploit scenario

1. Bob opens a deposit request under an affected policy.
2. Alice derives the request address, supplies Bob as the unchecked `owner`, signs as herself, and calls `reject_deposit_request`.
3. Bob cannot settle the rejected request and must cancel it to recover escrow, pay for a replacement, and remain vulnerable to repeated rejection while each replacement is pending.

Recommendation

Require direct core authorization for request decisions, such as a vault-administrator signature.

Define distinct policy contexts for approval and rejection instead of mapping them to ordinary request contexts.

Preserve owner-authenticated cancellation as the recovery path.

Acknowledgment 1.1

> Most "affected" policies are demo and out of scope, while `checkout` policy is production ready. Deposits on `checkout_policy` aren't asynchronous and expect transaction with both instruction: request and proccess
>
> — HyroTrader Team

Go back to Findings Summary

M4: Registry takeover blocks manager onboarding and child issuance

MEDIUM SEVERITY ISSUE

Impact:	Medium	**Likelihood:**	Medium
Type:	Access control	**Status:**	Fixed
Target:	programs/hyro_protocol/src/manager_registry.rs, programs/hyro_protocol/src/vault.rs, programs/policy_challenges/src/policy.rs		

Description

`initialize_manager_registry` assigns the singleton registry's administrator to any funded signer that initializes it first:

Listing 20. Excerpt from First caller becomes registry administrator

```
pub fn initialize_manager_registry(
    ctx: Context<InitializeManagerRegistry>,
) -> Result<()> {
    let registry = &mut ctx.accounts.registry;
    registry.admin = ctx.accounts.admin.key();
    registry.total_managers = 0;
    registry.total_aum = 0;
    registry.created_at = Clock::get()?.unix_timestamp as u64;
    Ok(())
}
```

Because no instruction rotates, closes, or reinitializes the administrator, that signer permanently controls manager registration and verification and can stop manager onboarding.

Child issuance requires the issuance signer to match both `registry.admin` and the parent vault administrator and requires a verified manager. These checks use one key for both administrator roles instead of collecting independent

approvals. A registry administrator can therefore issue children only for parents under the same key and can block issuance from parents controlled by other administrators, without gaining access to those parents' funds.

The takeover is possible only before initialization because `init` prevents replacement once the singleton address is occupied.

Exploit scenario

1. Alice, an unauthorized funded signer, calls `initialize_manager_registry` before the intended protocol operator and becomes the permanent registry administrator.
2. Alice refuses to register or verify the intended managers.
3. Bob, a parent vault administrator with a different key, cannot issue child vaults, and the operator cannot replace Alice.

Recommendation

Authenticate a deployment authority during singleton initialization and add a secure administrator rotation path. If the registry only verifies managers, remove the requirement that `registry.admin` equal the parent vault administrator. If both administrators must approve child issuance, authenticate two separate signers.

Fix 1.1

Registry initialization now authenticates the program upgrade authority, the current registry administrator can appoint a successor, and vault issuance no longer requires the registry administrator to equal the vault administrator.

Go back to Findings Summary

M5: Address collision can permanently block manager registration

MEDIUM SEVERITY ISSUE

Impact:	Medium	**Likelihood:**	Medium
Type:	Data validation	**Status:**	Fixed
Target:	programs/hyro_protocol/src/vault.rs, programs/hyro_protocol/src/manager_registry.rs		

Description

Root vaults derive their addresses from an unprefixed caller-supplied seed, while the global manager registry uses the fixed seed `manager_registry` under the same program:

Listing 21. Excerpt from Root vault uses its supplied string as the seed

```
382 #[account(
383     init,
384     payer = signer,
385     space = 8 + Vault::INIT_SPACE,
386     seeds = [seed.as_ref()],
387     bump,
388 )]
389 pub vault: Box<Account<'info, Vault>>,
```

Listing 22. Excerpt from Manager registry uses the fixed seed

```
51 #[account(
52     init,
53     payer = admin,
54     space = 8 + <ManagerRegistry as Space>::INIT_SPACE,
55     seeds = [b"manager_registry"],
56     bump,
57 )]
58 pub registry: Account<'info, ManagerRegistry>,
```

Before the registry is initialized, any caller who supplies the normal vault accounts and rent can call `initialize_vault` with `seed = "manager_registry"` and create a valid vault at the registry address without a protocol role. The registry initializer cannot overwrite or deserialize this vault, and no instruction can close it. Manager registration, verification, and child issuance then remain unavailable until the program is changed or redeployed.

Exploit scenario

1. Alice initializes a root vault with `seed = "manager_registry"` before the protocol initializes the registry.
2. Bob attempts to initialize the registry, but Alice's vault already occupies its singleton address, permanently blocking the registry lifecycle in the current program.

Recommendation

Domain-separate root vault addresses with a prefix such as `[b"vault", seed]`, update client derivations, and migrate existing addresses. Reject reserved singleton seeds during vault creation until migration is complete. Require authorization to initialize the manager registry.

Fix 1.1

Root vault PDAs are now namespaced, preventing collisions with the manager registry in future deployments.

Go back to Findings Summary

M6: Unauthorized policy setup can block vault operations

MEDIUM SEVERITY ISSUE

Impact:	Medium	**Likelihood:**	Medium
Type:	Access control	**Status:**	Acknowledged
Target:	programs/policy_limit_transfer/src/lib.rs, programs/policy_withdrawal_window/src/lib.rs		

Description

The limit-transfer and withdrawal-window initializers accept any funded signer and use an unchecked vault address only as a PDA seed, allowing an attacker to initialize the canonical policy state before the vault exists:

Listing 23. Excerpt from Limit policy derives state from an unchecked vault and payer

```
149 #[instruction(min: u64, max: u64)]
150 pub struct Initialize<'info> {
151     /// CHECK: This is the vault account used as seed for PDA derivation
152     vault: UncheckedAccount<'info>,
153
154     #[account(
155         init,
156         seeds = [vault.key().as_ref()],
157         bump,
158         payer = signer,
```

Listing 24. Excerpt from Window policy accepts an unchecked vault and signer

```
99 #[derive(Accounts)]
100 pub struct Initialize<'info> {
101     #[account(mut)]
102     pub authority: Signer<'info>,
103     /// CHECK: vault pubkey used only as PDA seed
104     pub vault: UncheckedAccount<'info>,
105     #[account(
106         init,
```

```
107         payer = authority,
108         space = 8 + WithdrawalWindowConfig::INIT_SPACE,
109         seeds = [vault.key().as_ref()],
110         bump,
111     )]
112     pub policy_account: Account<'info, WithdrawalWindowConfig>,
113     pub system_program: Program<'info, System>,
```

The `init` constraint then prevents the intended administrator from replacing the attacker's state. A limit policy with `min = u64::MAX` and `max = 0` rejects every parsable transaction amount, while a far-future `epoch_zero` prevents withdrawal processing. Neither policy provides a setter or close path, so the vault administrator must pause transactions, prepare another policy, and switch to it; withdrawal owners can cancel pending requests and recover their shares.

Exploit scenario

1. Alice, an attacker, initializes the policy state for Bob's planned vault before Bob creates it.
2. Alice sets impossible transfer bounds or a withdrawal epoch far in the future.
3. Bob cannot initialize the intended policy, and affected operations fail until he replaces the policy.

Recommendation

Deserialize the supplied vault and require its current administrator to sign both initializers. Add administrator-gated setters, require `min ⇐ max`, and bound the supported withdrawal epoch. Preserve policy replacement and owner cancellation as recovery mechanisms.

Acknowledgment 1.1

> Affected policies are mocks and our of audition scope. But thank you for information for future policies design

> — HyroTrader Team

Go back to Findings Summary

M7: Unrelated admin signatures bypass the withdrawal gate

MEDIUM SEVERITY ISSUE

Impact:	Medium	**Likelihood:**	Medium
Type:	Logic error	**Status:**	Acknowledged
Target:	lib/hyro_sdk/src/sigverify.rs, programs/policy_sig_gate/src/lib.rs		

Description

`has_signed_message` returns whether the expected public key signed an Ed25519 message but does not bind the message contents to the withdrawal request:

Listing 25. Excerpt from Signed message is reduced to a boolean

```
109 pub fn has_signed_message(
110     instructions_sysvar: Option<&AccountInfo>,
111     expected_signer: &Pubkey,
112 ) -> core::result::Result<bool, SigVerifyError> {
113     match find_signed_message(instructions_sysvar, expected_signer) {
114         Ok(_) => Ok(true),
115         Err(SigVerifyError::NoEd25519Instruction) |
    Err(SigVerifyError::SignerNotFound) => {
116             Ok(false)
117         }
118         Err(other) => Err(other),
119     }
```

The withdrawal policy treats this boolean as complete administrator approval:

Listing 26. Excerpt from Any administrator message approves withdrawal

```
58 let admin_signed = has_signed_message(vctx.instructions_sysvar(remaining),
   &vault.admin)
59     .map_err(|_| error!(ErrorCode::MissingInstructionsSysvar))?;
```

```
60 require!(admin_signed, ErrorCode::MissingAdminSignature);
61 Ok(())
```

Because the signed message contains no enforced vault, operation, requester, amount, expiry, or nonce, a requester who obtains packageable Ed25519 message bytes and a signature from the current administrator can reuse them to withdraw the requester's own escrowed shares without administrator consent. Rotation of the administrator invalidates signatures from the previous key.

Exploit scenario

1. Alice, a vault shareholder, obtains an unrelated message signed by Bob, the current administrator, in the format accepted by the verifier.
2. Alice packages that message and signature with her withdrawal request, causing the policy to approve withdrawal of her escrowed shares despite Bob never approving the request.

Recommendation

Define and verify a domain-separated approval message containing the vault, operation, requester, amount, and expiry. Include and consume a nonce to make each approval single-use. Replace the presence-only boolean API with verification that binds the signature to the complete withdrawal request.

Acknowledgment 1.1

> Affected policy is out of audition scope
>
> — HyroTrader Team

Go back to Findings Summary

M8: Unbounded oracle reports enable immediate share-price manipulation

MEDIUM SEVERITY ISSUE

Impact:	High	**Likelihood:**	Low
Type:	Data validation	**Status:**	Acknowledged
Target:	programs/hyro_protocol/src/funds.rs		

Description

`report_on_funds` writes the designated oracle's reported value directly to `offchain_balance` without limiting its deviation or delaying settlement after the change:

Listing 27. Excerpt from Oracle value directly updates off-chain balance

```
1718 match balance {
1719     BalanceUpdate::Delta(delta) => {
1720         ctx.accounts
1721             .vault
1722             .update_offchain_balance(BalanceUpdate::Delta(delta))?;
1723     }
1724     BalanceUpdate::NewValue(value) => {
1725         ctx.accounts
1726             .vault
1727             .update_offchain_balance(BalanceUpdate::NewValue(value))?;
1728     }
1729 }
```

Withdrawals immediately use the resulting NAV:

Listing 28. Excerpt from Withdrawal uses current NAV

```
1235 let assets = ctx.accounts.vault.convert_to_assets(shares, supply)?;
1236 require!(
1237     assets > 0,
1238     crate::vault::ErrorCode::ZeroAssetsRedeemed
1239 );
```

```
1240 require!(
1241     assets <= ctx.accounts.vault_token_account.amount,
1242     crate::vault::ErrorCode::InsufficientFunds
1243 );
```

A compromised designated oracle can therefore manipulate the share price when the active policy permits users to settle against the report. The limit-transfer and allow-any policies permit this flow; the checkout policy requires the vault administrator for reporting and withdrawals, and the challenge policy denies withdrawals. An inflated report lets the oracle redeem shares against overstated equity and drain available vault liquidity at other holders' expense, while a deflated report enables underpriced deposits before the reported equity is restored.

Exploit scenario

1. Alice controls the designated oracle and holds shares with a mature withdrawal request under a permissive settlement policy.
2. Alice reports inflated off-chain equity.
3. Alice settles at the resulting share price, selecting the report and redemption amount to drain the vault's available token balance while all ordinary policy and settlement checks pass.

Recommendation

Limit report deviations and require independent confirmation for unusual changes. Prevent immediate settlement after publication by binding requests to a confirmed report or settlement epoch. Record report freshness and define an authorized process for genuine large valuation changes.

Acknowledgment 1.1

> Implementation of any maximum change, delay, second confirmations is topic of policy design. System is flexible

> enough for such protection. For `policy_challenges` trust to oracle is design of the system.
>
> — HyroTrader Team

Go back to Findings Summary

L1: Unexecutable proposals have no rent-recovery path

LOW SEVERITY ISSUE

Impact:	Low	**Likelihood:**	Low
Type:	Code quality	**Status:**	Fixed
Target:	programs/hyro_protocol/src/vault.rs		

Description

`create_tx` charges the proposer to initialize a transaction account derived from `[vault, nonce]`, but only successful `execute_tx` closes the account and refunds its rent to the execution signer.

Listing 29. Excerpt from Proposer-funded transaction account

```
#[account(
    init,
    payer = signer,
    seeds = [vault.key().as_ref(), nonce.to_le_bytes().as_ref()],
    bump,
    space = transaction_space(accs.len(), data.len()))]
transaction: Box<Account<'info, Transaction>>,
```

Listing 30. Excerpt from Execution-only transaction closure

```
#[account(
    mut,
    close = signer,
    has_one = vault @ ErrorCode::TransactionVaultMismatch,
    seeds = [vault.key().as_ref(),
transaction.nonce.to_le_bytes().as_ref()],
    bump,
)]
transaction: Box<Account<'info, Transaction>>,
```

No cancellation or rent-reclamation instruction exists. If the target instruction is permanently invalid or the required authorization cannot be

obtained, the proposal cannot satisfy execution and permanently retains its rent while occupying that nonce. A valid proposal that participants abandon has the same result. A pause strands a proposal only until execution becomes possible again under a compatible policy.

Exploit scenario

1. Alice, a proposer authorized by Bob's vault policy, creates a transaction whose target instruction cannot succeed.
2. Bob, the vault administrator, abandons the proposal.
3. Neither Alice nor Bob can close the transaction without executing the target, so Alice's rent remains allocated and the proposal's nonce remains occupied, although other nonces remain available.

Recommendation

Add an authenticated cancellation instruction that closes a proposal without invoking its target. Record the original rent payer or another explicitly authorized refund destination at creation, and restrict the refund to that identity or a defined administrator policy. Allow proposals invalidated by policy changes to be canceled and reclaimed without permitting arbitrary callers to receive the rent.

Fix 1.1

New proposals can be canceled by their creator or the vault administrator, which fixes rent recovery in future deployments.

Go back to Findings Summary

L2: Challenge administrators can appoint child equity oracles

LOW SEVERITY ISSUE

Impact:	Low	**Likelihood:**	Medium
Type:	Access control	**Status:**	Fixed
Target:	programs/policy_challenges/src/policy.rs, programs/hyro_protocol/src/vault.rs, programs/hyro_protocol/src/funds.rs		

Description

`issue_child_vault` requires the signer to be both `ChallengeConfig.admin` and the parent vault administrator, then assigns that signer as the child vault administrator. The signer can therefore create or replace the child's `VaultOracle`, although the equity oracle is intended to be a separate reporting key.

Equity reporting accepts either the child vault's oracle or the parent vault's oracle:

Listing 31. Excerpt from Child and parent oracles are accepted

```
1665 let oracle_vault = ctx.accounts.vault_oracle.vault;
1666 let oracle_is_own = oracle_vault == ctx.accounts.vault.key();
1667 let oracle_is_parents = ctx.accounts.vault.parent_vault ==
     Some(oracle_vault);
1668 require!(
1669     oracle_is_own || oracle_is_parents,
1670     crate::vault::ErrorCode::UnauthorizedOracle
1671 );
```

The challenge administrator can appoint another key as the child oracle and publish child equity without the existing parent oracle. This preserves the

administrator's existing control over challenge status but removes independent authorization of the supporting equity record.

Exploit scenario

1. Alice, the challenge and parent vault administrator, issues a child vault and appoints her own key as its oracle.
2. Alice reports the child's equity without Bob, the parent vault oracle, signing the report.

Recommendation

Require challenge child vaults to use only the parent vault's oracle, and prevent the challenge administrator from rotating it.

Alternatively, change issuance so the challenge administrator does not automatically become the child vault administrator who can set its oracle.

Fix 1.1

Child vaults can no longer install independent oracle records, and their reports must use the parent vault's authenticated oracle record.

Go back to Findings Summary

L3: Unbound challenge registration blocks policy setup

LOW SEVERITY ISSUE

Impact:	Low	**Likelihood:**	Medium
Type:	Access control	**Status:**	Fixed
Target:	programs/policy_challenges/src/lib.rs, programs/policy_challenges/src/policy.rs		

Description

`register_challenge` authenticates the administrator of a parent configuration but accepts an unchecked `child_vault` and uses it to derive and initialize the policy-state PDA:

Listing 32. Excerpt from Unchecked child determines the policy-state address

```
261 /// CHECK: the child vault this challenge anchors. It need not exist yet —
    the
262 /// address is deterministic, so the challenge can be registered first.
    Used only
263 /// as the seed for `challenge`.
264 pub child_vault: UncheckedAccount<'info>,
265
266 /// The child vault's policy state account (`seeds = [child_vault]`).
267 #[account(
268     init,
269     payer = admin,
270     space = 8 + Challenge::INIT_SPACE,
271     seeds = [child_vault.key().as_ref()],
272     bump,
273 )]
274 pub challenge: Account<'info, Challenge>,
```

An administrator can therefore register an unrelated planned vault under their own configuration and occupy that vault's canonical policy-state address. The legitimate setup later fails at `init` because the existing state cannot be overwritten. The mismatched challenge cannot report equity for

the affected vault, but it blocks that vault's policy setup.

Exploit scenario

1. Alice, an administrator, registers Bob's planned vault address as a child of Alice's parent configuration.
2. Bob attempts to initialize the vault's policy state, but the initialization fails because Alice already occupies its canonical PDA.
3. Bob must change the planned vault or policy.

Recommendation

For an existing child, require `child.parent_vault == Some(config.vault)` and the expected child policy. For pre-registration, derive the child address from the authorized parent and supplied child seed instead of accepting an arbitrary address.

Fix 1.1

Challenge registration now binds the policy state to the child PDA derived from the authorized parent and `child_seed`. The out-of-scope generated Rust client omits `child_seed` and cannot encode a valid call.

Go back to Findings Summary

L4: Child and transaction addresses collide

LOW SEVERITY ISSUE

Impact:	Low	**Likelihood:**	Medium
Type:	Data validation	**Status:**	Fixed
Target:	programs/hyro_protocol/src/vault.rs		

Description

Transaction accounts derive from `[vault, nonce.to_le_bytes()]`, child vaults derive from `[parent_vault, seed]`, and authority PDAs derive from `[parent_vault]`, without domain prefixes.

For the same parent vault, an eight-byte child seed equal to a transaction nonce's little-endian bytes derives the transaction address, while an empty child seed derives the authority address. Because each account is created with `init`, the first creation occupies the address and blocks creation of the other account role. Callers must then choose another child seed or transaction nonce.

Exploit scenario

1. Alice, an authorized proposer, creates a transaction with a nonce whose eight-byte representation matches Bob's planned child seed.
2. The transaction occupies the shared address, so Bob, an administrator, must create the child vault with a different seed.

Recommendation

Add fixed domain prefixes to each derivation, such as `[b"tx", vault, nonce]` and `[b"child", parent_vault, seed]`. Update every corresponding derivation and migrate or recreate accounts that use the old addresses.

Fix 1.1

Child and transaction PDAs now use separate namespaces, preventing future address collisions. Existing children and proposals are not migrated to the new addresses.

Go back to Findings Summary

L5: Proposal execution forwards the executor's signature

LOW SEVERITY ISSUE

Impact:	Medium	**Likelihood:**	Low
Type:	Access control	**Status:**	Fixed
Target:	programs/hyro_protocol/src/vault.rs		

Description

A proposer controls the account metadata for a proposed CPI, including each account's `is_signer` flag. During `execute_tx`, the core sets the vault authority as a signer but preserves all other signer flags:

Listing 33. Excerpt from Executor signer flag is preserved

```
870 ix.accounts = ix
871     .accounts
872     .iter()
873     .map(|acc| {
874         let mut acc = acc.clone();
875         // Only the vault *authority* PDA is ever signed for. Note this is
    NOT
876         // `vault_share_signer` (which owns the vault token account and
    the share
877         // mint), so a proposed CPI can never mint shares or move the
    vault's
878         // primary token balance.
879         if acc.pubkey == ctx.accounts.vault.authority {
880             acc.is_signer = true;
881         }
882         acc
883     })
884     .collect();
```

When a proposal marks the executor as a signer, the executor's signature authorizing `execute_tx` is also forwarded to the target program. The target can therefore use that signature as the executor's personal or administrative authority, allowing the proposal to transfer the executor's assets or modify

configuration under the executor's control instead of acting only through the vault authority PDA. The executor must sign the transaction and supply each required writable account, but the resulting changes can persist after the executor removes the proposer from the manager role.

Exploit scenario

1. Alice, an authorized manager, proposes transferring the fee authority to herself and marks Bob, the current fee authority, as a required signer.
2. Bob executes the proposal as vault administrator, causing the fee program to accept the forwarded signature and transfer his authority to Alice.
3. Alice redirects fee recipients or replaces the calculator without another signature from Bob. Removing Alice as manager does not restore Bob's fee control.

Recommendation

Clear every signer flag except the vault authority PDA's flag before the CPI, and reject non-vault signer flags when proposals are created. Implement any required external co-signing through an explicit authorization flow.

Fix 1.1

Transaction creation now rejects non-vault signer metas, and execution removes forwarded signer privileges so only the vault authority PDA can sign the target invocation.

Go back to Findings Summary

L6: Invalid fee program configuration blocks fund operations

LOW SEVERITY ISSUE

Impact:	Low	**Likelihood:**	Medium
Type:	Configuration	**Status:**	Reported
Target:	programs/hyro_protocol/src/vault.rs, programs/hyro_protocol/src/funds.rs		

Description

`initialize_vault` stores the `fee_collection_program` public-key argument without checking that the account is executable or implements the expected fee interface:

Listing 34. Excerpt from Stored fee-program pointer

```
682 ctx.accounts.vault.policy_program = policy_program;
683 ctx.accounts.vault.fee_collection_program = fee_collection_program;
684 ctx.accounts.vault.seed = seed;
685 ctx.accounts.vault.authority = ctx.accounts.authority.key();
686 ctx.accounts.vault.admin = ctx.accounts.signer.key();
687 ctx.accounts.vault.share_mint = ctx.accounts.share_mint.key();
688 ctx.accounts.vault.underlying_mint = ctx.accounts.underlying_mint.key();
```

Deposit and withdrawal settlement, `use_funds`, `return_funds`, reporting, and fee claims invoke the stored address. An incorrect or incompatible address therefore blocks these operations. Because no administrator instruction can replace `Vault::fee_collection_program`, and `set_fee_calc_program` changes only the calculator behind a valid fee ledger, recovery requires a core-program change or migration to a correctly configured vault.

A user can submit a deposit request because that operation does not invoke the fee program, but settlement subsequently fails. The user can cancel the

request and recover the escrowed assets without a fee CPI.

Exploit scenario

1. Bob, an administrator, initializes a vault with a mistyped fee-program address.
2. Alice submits a deposit request, but settlement fails when the core program invokes that address.
3. Alice cancels the request and recovers her escrowed assets, while Bob cannot repair the stored pointer through an existing administrator instruction.

Recommendation

Require an executable, supported fee-program account during initialization.

Add an administrator-authenticated replacement or migration path that preserves existing fee liabilities and recipient state.

Update 1.1

Program selections now require executable accounts, but they do not verify a compatible interface, funded children still copy the parent fee pointer, and unusable programs with positive liabilities cannot be replaced.

Go back to Findings Summary

L7: Invalid calculator configuration halts fee-charging operations

LOW SEVERITY ISSUE

Impact:	Low	**Likelihood:**	Low
Type:	Data validation	**Status:**	Reported
Target:	programs/fee_collection/src/lib.rs		

Description

Fee-ledger initialization and `set_fee_calc_program` store a calculator public key without verifying that it identifies an executable program implementing the fee-calculation interface. `set_fee_calc_program` directly assigns the key, and `UpdateFeeConfig` provides no calculator account to validate:

Listing 35. Excerpt from Calculator identifier update

```
319 pub fn set_fee_calc_program(ctx: Context<UpdateFeeConfig>, program:
    Pubkey) -> Result<()> {
320     ctx.accounts.vault_fees.fee_calc_program = program;
321     ctx.accounts.vault_fees.last_update = Clock::get()?.unix_timestamp;
322     Ok(())
323 }
```

Fee-charging operations require the supplied calculator account to match the stored identifier:

Listing 36. Excerpt from Stored calculator enforced at charge time

```
352 // Verify fee calc program matches stored program
353 require!(
354     ctx.accounts.fee_calc_program.key() == fees.fee_calc_program,
355     ErrorCode::InvalidFeeCalcProgram
356 );
```

The dispatcher then invokes that address through the fee-calculation ABI. A non-executable address cannot be invoked, while an unrelated executable program may reject the instruction or return incompatible data. An invalid configuration therefore causes deposits, withdrawals, fund movements, and reports to fail when they call `invoke_charge_fees`. The outage persists until the stored fee authority restores a compatible calculator; if that authority was not rotated, recovery may depend on an outgoing administrator as described in H2.

Exploit scenario

1. Alice, the fee authority, accidentally sets the calculator identifier to a non-program address.
2. Bob, a vault user, submits an otherwise valid fee-charging operation, but the dispatcher cannot invoke the configured address and the transaction reverts.
3. Bob cannot complete the operation until Alice restores a valid calculator.

Recommendation

Require both initialization and `set_fee_calc_program` to receive an account matching the proposed calculator identifier and verify that it is executable. Validate compatibility with the fee-calculation ABI through an explicit interface or capability check. Maintain an authenticated recovery path and document which fee authority can use it after vault administration changes.

Update 1.1

Fee calculator accounts must now be executable, but no supported-interface check exists and the supplied Rust initializer omits the newly required calculator-program account.

Go back to Findings Summary

L8: Unchecked policy selection interrupts vault operations

LOW SEVERITY ISSUE

Impact:	Low	**Likelihood:**	Low
Type:	Configuration	**Status:**	Acknowledged
Target:	programs/hyro_protocol/src/vault.rs		

Description

`initialize_vault` accepts an unchecked `policy_program` public key, and `issue_child_vault` stores the key of an unchecked `child_policy` account:

Listing 37. Excerpt from Unchecked child policy account

```
643 /// CHECK: child policy
644 pub child_policy: UncheckedAccount<'info>,
```

Neither path verifies that the selected address is executable or implements the required policy interface. Parent-policy authorization during child issuance does not validate the child's policy implementation. Policy replacement checks executability, but it does not verify compatibility with the expected `validate` discriminator and account contract:

Listing 38. Excerpt from Policy replacement executable check

```
1146 #[account(
1147     constraint = new_policy.executable @
     ErrorCode::PolicyProgramNotExecutable
1148 )]
1149 pub new_policy: UncheckedAccount<'info>,
```

A non-executable or incompatible policy causes every vault operation that invokes it to fail. The vault administrator can recover without approval from the broken policy by pausing transactions, replacing the policy, provisioning

the replacement's required state, and resuming transactions. If no configuration account exists, the pause instruction creates it. Deposit and withdrawal owners can also cancel pending requests without invoking the policy. Recovery therefore depends on the administrator key remaining available.

Exploit scenario

1. Bob, a parent administrator who satisfies the registry and parent-policy issuance checks, issues a child vault with a non-executable `child_policy` address.
2. Alice, the child's intended operator, cannot complete operations that invoke the policy.
3. Bob pauses transactions, replaces the policy through the administrator instructions, provisions any required policy state, and resumes operations after the child's other prerequisites, including share and fee accounts, are ready.

Recommendation

Validate executable policy accounts during root-vault creation, child-vault issuance, and policy replacement. Verify a supported policy interface or version, including the expected instruction and account contract. Preserve policy-independent administrator recovery and explicitly require provisioning all state needed by a replacement policy.

Acknowledgment 1.1

> Not an issue with availability of `update_policy_program` instruction
>
> — HyroTrader Team

Go back to Findings Summary

L9: Delayed approvals authorize recycled proposals

LOW SEVERITY ISSUE

Impact:	Medium	**Likelihood:**	Low
Type:	Logic error	**Status:**	Acknowledged
Target:	programs/policy_multisig/src/lib.rs, programs/hyro_protocol/src/vault.rs		

Description

`approve` finds a pending proposal only by its transaction public key and records the signing owner's approval without authenticating the proposal contents or generation:

Listing 39. Excerpt from Address-only approval lookup

```
356 pub fn approve(ctx: Context<Approve>, transaction: Pubkey) -> Result<()> {
357     let now = Clock::get()?.unix_timestamp;
358     let multisig = &mut ctx.accounts.policy_account;
359
360     let index = owner_index(multisig, ctx.accounts.owner.key)?;
361     let pending = multisig
362         .pending
363         .iter_mut()
364         .find(|pending| pending.transaction == transaction)
365         .ok_or(error!(ErrorCode::NoPendingTransaction))?;
```

The core derives a transaction address from the vault and a caller-chosen nonce, then closes the account after execution. An authorized proposer can therefore reuse the address for different instruction content with fresh approval flags. A valid approval transaction that is signed for the original proposal but processed after this reuse applies to the replacement proposal because it binds only to the recycled address.

Exploitation requires sufficient approvals to execute the original proposal

without the delayed vote, reuse of its address while the delayed Solana transaction remains valid, and an unchanged owner set that still recognizes the signer. The delayed approval can then help satisfy the threshold for unreviewed instructions and expose assets controlled by the vault-authority PDA.

Exploit scenario

1. Alice, a malicious owner of a two-of-three vault multisig, and a third owner approve proposal A at address P.
2. Bob, another owner, signs a standalone approval for A, but Alice delays its submission through a relay she controls.
3. Alice executes A, which closes P, creates malicious proposal B at P with the same vault and nonce, and approves B.
4. Alice submits Bob's still-valid, unprocessed approval; it applies to B and reaches the two-vote threshold, allowing Alice to execute instructions Bob never reviewed.

Recommendation

Bind every approval to a unique proposal generation or to a digest covering the vault, target program, instruction data, and account metas, and validate that binding against the current transaction before recording the vote.

Derive proposal addresses from a monotonically increasing per-vault identifier if they must never be reused.

Preserve the authenticated proposal binding when clearing approvals, and authenticate any creation timestamp included in that binding.

Acknowledgment 1.1

> `policy_multisig` is demo policy for documentation, out of audition scope

> — HyroTrader Team

Go back to Findings Summary

L10: One multisig owner can veto threshold-approved proposals

LOW SEVERITY ISSUE

Impact:	Low	**Likelihood:**	Medium
Type:	Logic error	**Status:**	Acknowledged
Target:	programs/policy_multisig/src/lib.rs		

Description

`cancel` allows any current multisig owner to remove a pending proposal by transaction address, regardless of its expiry or collected approvals:

Listing 40. Excerpt from Single-owner proposal removal

```
387 owner_index(multisig, ctx.accounts.owner.key)?;
388
389 let index = multisig
390     .pending
391     .iter()
392     .position(|pending| pending.transaction == transaction)
393     .ok_or(error!(ErrorCode::NoPendingTransaction))?;
394 multisig.pending.remove(index);
```

In an M-of-N configuration where M is below N, one malicious or compromised owner can therefore delete a live proposal after the other owners reach the approval threshold. Execution then fails because the pending entry no longer exists. The core transaction account remains open, so replacement requires an available transaction address and a new round of approvals. Repeated cancellation gives one owner a liveness veto until the vault administrator removes that owner through `update_owners`, which also clears pending proposals.

Exploit scenario

1. Alice, a malicious owner in a two-of-three multisig, withholds her approval.
2. Bob, an honest owner, and the third owner approve a legitimate transaction.
3. Alice calls `cancel` with the transaction address before Bob executes it, deleting the threshold-approved proposal and causing Bob's execution to fail.
4. Alice repeats this against replacement proposals until the vault administrator removes her.

Recommendation

Require threshold authorization to cancel a live proposal, or prevent a single owner from cancelling a proposal after it reaches the approval threshold. Permit cleanup of expired proposals without granting one owner cancellation authority over live proposals. Correct the comment that equates cancellation with withholding a vote.

Acknowledgment 1.1

> `policy_multisig` is demo policy for documentation, out of audition scope
>
> — HyroTrader Team

Go back to Findings Summary

L11: Direct returns can be omitted from NAV if the oracle reports first

LOW SEVERITY ISSUE

Impact:	Medium	**Likelihood:**	Low
Type:	Logic error	**Status:**	Fixed
Target:	programs/hyro_protocol/src/vault.rs, programs/hyro_protocol/src/funds.rs		

Description

A direct venue return to the vault's primary token account is omitted from NAV if the oracle reports the reduced off-chain balance before reconciliation.

Reconciliation credits an actual-custody surplus only up to the recorded off-chain balance, preventing unexplained token donations from changing the share price:

Listing 41. Excerpt from Donation-resistant reconciliation

```
174 if balance_diff > 0 {
175     let surplus = balance_diff as u128;
176     // Cap the reconciliation at what off-chain capital can account for.
177     let explained = (stored_offchain as u128).min(surplus);
178     if explained > 0 {
179         let decrease =
180             u64::try_from(explained).map_err(|_|
    error!(ErrorCode::MathOverflow))?;
```

The supported `return_funds` path avoids this ordering dependency by transferring the tokens, crediting the on-chain balance, and retiring the corresponding off-chain balance atomically:

Listing 42. Excerpt from Atomic return bookkeeping

```
1634 let retired = assets.min(ctx.accounts.vault.offchain_balance);
1635 ctx.accounts
1636     .vault
1637     .update_onchain_balance(BalanceUpdate::Delta(
1638         i64::try_from(assets).map_err(|_| error!(
     crate::vault::ErrorCode::MathOverflow))?,
1639     ))?;
1640 ctx.accounts
1641     .vault
1642     .update_offchain_balance(BalanceUpdate::Delta(
1643         -(i64::try_from(retired)
1644             .map_err(|_| error!(
     crate::vault::ErrorCode::MathOverflow))?),
1645     ))?;
```

For a direct return, `report_on_funds` does not reconcile the receipt. Once the oracle reduces the recorded venue balance, later reconciliation cannot attribute the custody surplus to off-chain capital and excludes it from NAV. The tokens remain in custody, but shares are underpriced and bookkeeping prevents operations from debiting the full physical balance. No standalone instruction attributes a receipt after it becomes unexplained.

Exploit scenario

1. Bob, a vault operator holding Alice's deposit, directly returns `500,000` venue units to a vault that records `50,000` units on-chain and `500,000` units off-chain.
2. Bob's oracle truthfully reports a zero venue balance before reconciliation, reducing recorded NAV from `550,000` to `50,000`.
3. Reconciliation sees `500,000` excess custody units and no recorded off-chain capital, so it credits none of them.
4. Alice's shares remain priced against `50,000` units, and moving or withdrawing the full `550,000` units fails when bookkeeping debits the recorded on-chain balance.

Recommendation

Enforce and document a return sequence that routes venue withdrawals through authority custody, calls `return_funds`, and then reports the new venue balance.

If direct returns to primary custody must remain supported, add an authenticated attribution or standalone reconciliation instruction that credits legitimate receipts without accepting arbitrary donations.

Fix 1.1

The new administrator-only `attribute_donation` instruction can reconcile unrecorded custody tokens into the vault's on-chain balance without exceeding the physical token balance.

Go back to Findings Summary

L12: Request replacement bypasses rejection

LOW SEVERITY ISSUE

Impact:	Low	**Likelihood:**	Low
Type:	Logic error	**Status:**	Acknowledged
Target:	programs/hyro_protocol/src/funds.rs		

Description

Settlement rejects only `Rejected` requests, so both `Pending` and `Approved` deposit and withdrawal requests can settle without approval:

Listing 43. Excerpt from Settlement rejection check

```
855 require!(
856     ctx.accounts.deposit_request.status != RequestStatus::Rejected,
857     crate::vault::ErrorCode::RequestIsRejected
858 );
```

A rejection exists only in the request account. The owner-signed `cancel_deposit_request` and `cancel_withdrawal_request` handlers return the escrowed amount and close that account regardless of its status or policy. A replacement request starts at `Pending` and retains no record of the prior rejection:

Listing 44. Excerpt from Replacement request initialization

```
805 let dr = &mut ctx.accounts.deposit_request;
806 dr.vault = ctx.accounts.vault.key();
807 dr.owner = ctx.accounts.signer.key();
808 dr.assets = assets;
809 dr.requested_at = Clock::get()?.unix_timestamp;
810 dr.bump = ctx.bumps.deposit_request;
811 dr.status = RequestStatus::Pending;
```

The requester can therefore replace a rejected request and settle the replacement, bypassing a persistent admission veto even under `policy_owners`. Independent policy credentials, withdrawal delays, escrow requirements, and vault pauses remain enforced.

Exploit scenario

1. Bob, a vault operator using `policy_owners`, rejects Alice's deposit request to deny her admission.
2. Alice, the request owner, cancels the request and recovers her escrowed assets.
3. Alice creates a replacement that starts at `Pending` and settles it without approval, because Bob's rejection does not apply to the replacement.

Recommendation

Define whether rejection applies to a single request or to the owner's admission.

Persist owner-level denials outside closable request accounts when rejection must prohibit replacement requests. Require approval at settlement when approval authorizes entry, and bind each approval to the expected request version and amount. Preserve the owner's ability to recover unprocessed escrow, and update the status documentation to match the enforced lifecycle.

Acknowledgment 1.1

> Design the system to support synchronous operations. The policy may treat all pending transactions as approved for deposits/withdrawals in case of any deposit/withdrawal are allowed.
>
> — HyroTrader Team

Go back to Findings Summary

L13: Retired administrators retain manager access

LOW SEVERITY ISSUE

Impact:	Medium	**Likelihood:**	Low
Type:	Access control	**Status:**	Acknowledged
Target:	programs/policy_manager_access/src/lib.rs		

Description

Rotating `Vault.admin` does not invalidate the separate manager authorization set because `set_vault_admin` updates only the vault field:

Listing 45. Excerpt from Vault administrator rotation

```
1278 pub fn set_vault_admin(ctx: Context<UpdateVaultStatus>, new_admin:
     Pubkey) -> Result<()> {
1279     require!(new_admin != Pubkey::default(), ErrorCode::InvalidAdmin);
1280     ctx.accounts.vault.admin = new_admin;
1281     Ok(())
1282 }
```

`initialize_managers` and `update_managers` authenticate the current administrator but do not bind manager grants to that administrator or an authorization epoch. `require_authorized_with` therefore continues to accept any retained manager after the handover:

Listing 46. Excerpt from Administrator-or-manager authorization

```
286 if *signer == vault.admin {
287     return Ok(());
288 }
289 let policy = load_managers(policy_account)?;
290 require!(
291     policy.managers.contains(signer),
292     ErrorCode::UnregisteredManager
293 );
```

```
294 Ok(())
```

Because creation, execution, and custody movement use this predicate, a former administrator retained as a manager can call `use_funds` and execute a transfer through `execute_tx`. A manager set initialized while another policy is active can likewise become effective after a later policy switch because activation neither inspects nor resets the incoming policy state.

The successor can derive the deterministic `[vault]` manager account, inspect or overwrite its entries, change the active policy, and revoke managers without their cooperation. However, the old grants remain effective until the successor takes one of these actions.

Exploit scenario

1. Alice, the departing vault administrator and a listed manager, calls `set_vault_admin` to appoint Bob.
2. Alice uses her retained authorization before Bob updates the manager set to move tokens into authority-controlled custody, create a token-transfer proposal, and execute it through the vault authority.

Recommendation

Bind manager grants to the current administrator identity or a monotonic authorization epoch, and reject stale grants until the current administrator explicitly confirms them. Require policy activation to inspect and explicitly confirm or reset existing manager state. Expose the deterministic manager account and its entries during handover, permit obsolete state to be closed or reset, and preserve the current administrator's ability to revoke managers without their cooperation.

Acknowledgment 1.1

> `policy_manager_access` is demo policy for documentation, out of

> audition scope
>
> — HyroTrader Team

Go back to Findings Summary

L14: Retired and active policy binaries use the same address

LOW SEVERITY ISSUE

Impact:	Low	**Likelihood:**	Low
Type:	Logic error	**Status:**	Acknowledged
Target:	programs/policy_admin_only/src/lib.rs, programs/policy_manager_access/src/lib.rs, Cargo.toml		

Description

The active `policy_manager_access` program and retired `policy_admin_only` program declare the same on-chain address:

Listing 47. Excerpt from Active policy address

```
33 declare_id!("HyroXELvGkzBwgKz2Fsz2XEMDM6TYupZZ1A7qAtEM9Qs");
```

Listing 48. Excerpt from Retired policy address

```
46 declare_id!("HyroXELvGkzBwgKz2Fsz2XEMDM6TYupZZ1A7qAtEM9Qs");
```

Cargo's wildcard workspace still includes the retired crate, so a Cargo-based build can produce both artifacts even though Anchor's deployment list selects only `policy_manager_access`. If an operator with deployment or upgrade authority selects the retired binary for the shared address, vaults configured for the manager policy receive `Deprecated` from `validate` instead of administrator or registered-manager authorization, interrupting policy-gated operations until the active binary or another executable policy is selected.

Exploit scenario

1. Alice, a deployment operator, builds the Cargo workspace and mistakenly deploys the retired `policy_admin_only` artifact to the shared address.
2. Bob's vault uses that address for its manager policy, so its next policy-gated operation fails with `Deprecated`.

Recommendation

Remove the retired crate or exclude it from every deployable workspace. Build release artifacts from an explicit allowlist, and verify each artifact's program name and address before deployment or upgrade.

Acknowledgment 1.1

> `policy_manager_access` is demo policy for documentation, out of audition scope
>
> — HyroTrader Team

Go back to Findings Summary

L15: Unnecessary writable accounts couple independent vaults

LOW SEVERITY ISSUE

Impact:	Low	**Likelihood:**	Medium
Type:	Code quality	**Status:**	Reported
Target:	programs/hyro_protocol/src/vault.rs, programs/hyro_protocol/src/funds.rs		

Description

Core vault and fund contexts require write access to shared accounts that the selected operations do not modify, causing otherwise independent transactions to contend on Solana account locks.

`require_policy_account` accepts either a canonical per-vault policy PDA or the policy program address for stateless policies, but creation and execution contexts unconditionally mark `policy_account` mutable. Fund request, settlement, movement, and reporting contexts apply the same requirement, and `invoke_validate_with` propagates the account's writability into the policy CPI. Stateless vault transactions therefore request a write lock on the shared policy program address.

Listing 49. Excerpt from Unconditional policy-state write lock

```
519 /// CHECK: policy state account; validated against the canonical policy
    PDA in
520 /// the handler (AUDIT I-02). Mutable so stateful policies can clear
    approvals.
521 #[account(mut)]
522 policy_account: AccountInfo<'info>,
```

`ProcessDeposits`, `ProcessWithdrawals`, `UseFunds`, `ReturnFunds`, `RecalculateBalance`, and `ClaimFees` also mark the underlying mint mutable. The

live operations only transfer underlying tokens or mint and burn the separate share mint, so the underlying mint remains unchanged. `RecalculateBalance` has no exposed standalone instruction, but its unused context contains the same unnecessary declaration. The five live paths consequently couple vaults that share an underlying asset, regardless of their policy.

Listing 50. Excerpt from Underlying-mint write lock

```
95 #[account(mut, address = vault.underlying_mint @
   crate::ErrorCode::InvalidUnderlyingMint)]
96 pub underlying_mint: Account<'info, Mint>,
```

These unnecessary write locks prevent concurrent execution of conflicting transactions and create avoidable scheduling contention under load, although transactions can still execute sequentially within the same slot.

Exploit scenario

1. Alice and Bob submit settlement transactions for independent vaults backed by the same underlying mint.
2. Both transactions request a write lock on the shared mint and cannot execute concurrently, although their vault and token accounts differ.
3. If both vaults use stateless policies, their transactions also contend on the writable policy program account.

Recommendation

Remove `mut` from all six underlying-mint declarations while retaining their address constraints.

Permit read-only stateless policy accounts, require writable policy state only for policies and operations that modify it, and preserve the canonical policy-state ABI.

Regenerate and update instruction builders so callers request read-only

account metas after deploying program constraints that accept them.

Update 1.1

Underlying-mint accounts are now read-only, but stateless policy marker accounts remain writable and still create a shared write lock.

Go back to Findings Summary

L16: Transfers above the signed limit can reverse balance updates

LOW SEVERITY ISSUE

Impact:	Low	**Likelihood:**	Low
Type:	Arithmetics	**Status:**	Fixed
Target:	programs/hyro_protocol/src/funds.rs, programs/hyro_protocol/src/vault.rs		

Description

Several vault accounting paths cast `u64` token amounts to `i64` without checking that they do not exceed `i64::MAX`.

The withdrawal path transfers the unsigned amount before constructing its signed debit:

Listing 51. Excerpt from Unchecked withdrawal debit

```
1320 ctx.accounts
1321     .vault
1322     .update_onchain_balance(BalanceUpdate::Delta(-(assets as i64)))?;
```

For an amount above `2^63 - 1`, the cast produces a negative value, so negating it creates a positive balance delta; at exactly `2^63`, negating `i64::MIN` can overflow.

The deposit credit and both `use_funds` balance updates contain the same unchecked conversion.

Because `apply_delta` validates only the converted value, later range checks cannot detect the reversed sign.

The NAV constraint rejects most oversized transactions, but fee liabilities can

keep NAV within the `i64` range while custody exceeds it.

A successful oversized withdrawal therefore sends tokens out while increasing the recorded balance and corrupting vault accounting.

The threshold applies to raw units: it is approximately 9.22 billion tokens for a nine-decimal mint and 9.22 tokens for an eighteen-decimal mint.

Exploit scenario

Let `T = 2^63 - 1` and `V = 1,000,000`.

1. Alice, who controls a high-raw-supply mint, the vault, its fee recipients, its fractional-fee configuration, and a policy admitting every operation, deposits `T - V` and then `V` with a 100% manager fee, producing `T` units of recorded custody, `T` units of fees, zero NAV, and `2T - V` shares.
2. Alice claims the `T` fee, temporarily sets fees to zero, moves those tokens into authority custody, returns `T` to the vault, directly donates another `V`, and sets the withdrawal LP fee to 100%; recorded custody and NAV equal `T`, while actual custody equals `T + V`.
3. Alice redeems all shares and receives `T + 499,999`, which exceeds `i64::MAX` but remains below actual custody; fee booking leaves the representable NAV value `-499,999`.
4. The transfer leaves `500,001` actual units in custody, but the inverted debit raises the recorded balance to `18,446,744,073,709,051,617` and recorded NAV to `9,223,372,036,853,775,811`.

Recommendation

Convert every unsigned amount with `i64::try_from` before constructing a signed delta, and return `MathOverflow` when conversion fails.

Use checked unsigned credits and debits wherever a signed intermediate is unnecessary.

Fix 1.1

Token amounts now use checked conversions to `i64` before signed balance updates, so values above `i64::MAX` fail instead of wrapping.

Go back to Findings Summary

L17: One multisig owner can disable withdrawals

LOW SEVERITY ISSUE

Impact:	Medium	**Likelihood:**	Low
Type:	Access control	**Status:**	Acknowledged
Target:	programs/policy_multisig/src/lib.rs		

Description

The `UseFunds` and `ReturnFunds` policy arms authorize any signer in `multisig.owners` without enforcing `multisig.threshold`; only the `Execution` arm counts approvals against that threshold.

Listing 52. Excerpt from Threshold for proposed execution

```
198 let signature_count = count_signatures(&pending.signatures)?;
199 require!(
200     signature_count >= multisig.threshold,
201     ErrorCode::NotEnoughSigners
202 );
```

Listing 53. Excerpt from Single-owner custody authorization

```
211 ValidateContext::UseFunds(use_funds_ctx) => {
212     use_funds_ctx.validate_accounts(ctx.remaining_accounts)?;
213     let policy_account =
    use_funds_ctx.policy_account(ctx.remaining_accounts);
214     require_policy_account(&vault_key, policy_account, &crate::ID)?; //
    AUDIT C-02
215     let sender = use_funds_ctx.signer(ctx.remaining_accounts)?; // AUDIT
    C-02
216     let multisig = load_multisig(policy_account)?;
217     owner_index(&multisig, &sender.key())?;
218     Ok(())
219 }
```

`use_funds` transfers tokens from `vault_token_account` to

`authority_token_account`, so one owner can remove all liquid redemption capacity without the other owners' approval. Withdrawal settlement then fails while primary custody lacks the requested amount.

Listing 54. Excerpt from Withdrawal liquidity requirement

```
1240 require!(
1241     assets <= ctx.accounts.vault_token_account.amount,
1242     crate::vault::ErrorCode::InsufficientFunds
1243 );
```

An owner can restore the tokens through `return_funds`, but the same owner can repeatedly disrupt withdrawals. Each movement also incurs the separately reported gross-neutral transfer fees when those fees are configured.

Exploit scenario

1. Alice, one owner of a two-of-three multisig vault, calls `use_funds` for the full vault token account balance without another approval.
2. Bob, an honest owner processing a positive withdrawal, cannot settle it until he returns the tokens.
3. Alice repeats the movement, repeatedly disabling withdrawals and causing any configured movement fees to accrue on each cycle.

Recommendation

Require threshold approval for `UseFunds` and `ReturnFunds`, and bind each approval to the vault, operation, amount, and replay identifier.

If unilateral liquidity management is required, enforce an explicit limited allowance and document that the execution threshold does not protect redemption liquidity. Preserve the independent oracle and administrator authorization checks.

Acknowledgment 1.1

> `policy_multisig` is demo policy for documentation, out of audition scope
>
> — HyroTrader Team

Go back to Findings Summary

L18: Single-step handovers can lose administrative access

LOW SEVERITY ISSUE

Impact:	Low	**Likelihood:**	Low
Type:	Access control	**Status:**	Fixed
Target:	programs/hyro_protocol/src/vault.rs, programs/fee_collection_all_in_one/src/lib.rs		

Description

`set_vault_admin` and `fee_collection_all_in_one::update_authority` accept any non-default public key and immediately replace the active authority without requiring the successor to accept the role:

Listing 55. Excerpt from Immediate vault administrator replacement

```
1278 pub fn set_vault_admin(ctx: Context<UpdateVaultStatus>, new_admin:
     Pubkey) -> Result<()> {
1279     require!(new_admin != Pubkey::default(), ErrorCode::InvalidAdmin);
1280     ctx.accounts.vault.admin = new_admin;
1281     Ok(())
1282 }
```

Listing 56. Excerpt from Immediate calculator authority replacement

```
634 pub fn update_authority(ctx: Context<UpdateConfig>, new_authority: Pubkey)
    -> Result<()> {
635     require_keys_neq!(new_authority, Pubkey::default(),
    ErrorCode::InvalidConfig);
636     ctx.accounts.config_account.authority = new_authority;
637     Ok(())
```

Subsequent vault status and policy changes require the stored administrator to sign through `UpdateVaultStatus` and `UpdateVaultPolicy`. Calculator configuration changes similarly require the stored authority through

`UpdateConfig`; initialization cannot replace an existing configuration, and `migrate_vault_all_in_one_config` does not change an already-current account. Assigning an unintended or unusable key therefore prevents the former authority from reversing the transfer and permanently removes access to these administrative operations unless a separately authenticated recovery mechanism exists. Existing authorized vault operations and fee calculation can continue, and an independent fee-ledger authority can select another calculator through `set_fee_calc_program`, but it cannot recover the original calculator configuration authority.

Exploit scenario

1. Bob, a vault administrator, intends to transfer control to Alice, the incoming administrator, but submits a non-default key that Alice does not control.
2. The transfer takes effect immediately, so neither Bob nor Alice can authorize later vault status or policy changes.
3. The same error during a calculator-authority transfer prevents later updates to that calculator's configuration.

Recommendation

Implement a two-step handover in which the current authority records a pending successor, the successor signs to accept the role, and the current authority can cancel an unaccepted proposal. Emit dedicated events for handover proposals, cancellations, and acceptances. Add a separately authenticated recovery mechanism if recovery from loss of the current authority is part of the trust model.

Fix 1.1

Vault administrator handovers now require nomination and acceptance, and fee configuration authenticates the live vault administrator. The out-of-

scope `set-vault-admin.ts` operational script still calls the removed one-step instruction.

Go back to Findings Summary

L19: Exchange debt cannot be reflected in NAV

LOW SEVERITY ISSUE

Impact:	Medium	**Likelihood:**	Low
Type:	Data validation	**Status:**	Reported
Target:	programs/hyro_protocol/src/vault.rs		

Description

The vault cannot record off-chain liabilities because its off-chain balance and `BalanceUpdate::NewValue` are unsigned, although the aggregate `net_balance` is signed:

Listing 57. Excerpt from Stored balance types

```
93 pub onchain_balance: u64,
94 pub offchain_balance: u64,
95 pub total_balance: i64,
96 pub obtained_fees: u64,
```

A negative delta that exceeds the stored off-chain balance returns `BalanceUnderflow` instead of recording the deficit:

Listing 58. Excerpt from Negative-delta checked subtraction

```
322 current
323     .checked_sub(delta.unsigned_abs())
324     .ok_or_else(|| error!(ErrorCode::BalanceUnderflow))
```

Consequently, `report_on_funds` either rejects a report of negative exchange equity and preserves the previous valuation or accepts zero and omits the liability. Because `net_balance` deducts only fee liabilities from unsigned asset balances, it cannot include this debt.

If settlement remains enabled and custody holds sufficient tokens, the overstated NAV overvalues withdrawals and reduces shares minted for deposits. This requires an external strategy loss exceeding its posted capital that creates debt collectible from the vault; ordinary losses that leave nonnegative equity remain representable.

Exploit scenario

1. Alice deposits into a vault, and a policy-authorized manager places vault capital in an external strategy that incurs a collectible loss exceeding the posted capital.
2. Bob, the honest oracle operator, cannot report the resulting negative equity because the update fails when the loss exceeds the stored off-chain balance, while reporting zero omits the deficit.
3. If withdrawals remain authorized and custody can pay them, Alice redeems against the overstated NAV, leaving the unrecorded liability with the vault and its remaining holders.

Recommendation

Represent off-chain net equity with a signed balance or add an explicit off-chain liability deducted from NAV. Update absolute and delta reports, conversions, and SDK encodings consistently, and explicitly migrate stored accounts.

If negative exchange equity is outside the supported asset model, enforce that constraint and suspend price-sensitive settlement whenever a liability cannot be represented while preserving the distinction between a zero balance and a known deficit.

Update 1.1

Off-chain balances are now signed, but a truthful loss report is still rejected when existing fee liabilities exceed the resulting nonnegative gross balance.

Go back to Findings Summary

L20: Fixed virtual assets distort final redemptions

LOW SEVERITY ISSUE

Impact:	Low	**Likelihood:**	Medium
Type:	Logic error	**Status:**	Reported
Target:	programs/hyro_protocol/src/vault.rs		

Description

The share conversion adds fixed virtual assets and shares of 1,000,000 raw units regardless of the underlying mint's decimals, so redemption returns `floor(shares * (NAV + 1_000_000) / (supply + 1_000_000))` instead of the holder's unadjusted pro-rata NAV:

Listing 59. Excerpt from Virtualized redemption ratio

```
297 let numerator = (shares as u128)
298     .checked_mul(
299         total_assets
300             .checked_add(VIRTUAL_ASSETS)
301             .ok_or(ErrorCode::MathOverflow)?,
302     )
303     .ok_or(ErrorCode::MathOverflow)?;
304 let denominator = (supply as u128)
305     .checked_add(VIRTUAL_SHARES)
306     .ok_or(ErrorCode::MathOverflow)?;
```

For a full redemption, the pre-flooring difference from NAV is `1_000_000 * (supply - NAV) / (supply + 1_000_000)`. If supply exceeds NAV, the payout exceeds NAV by less than 1,000,000 raw units and can consume tokens owed as accrued fees when sufficient token liquidity exists. If NAV exceeds supply, burning all real shares leaves the virtual-share fraction of subsequent profit as residual NAV.

The deposit path rejects a zero-supply vault with positive residual NAV:

Listing 60. Excerpt from Zero-supply deposit condition

```
920 if ctx.accounts.share_mint.supply == 0 {
921     require!(
922         ctx.accounts.vault.total_balance == 0,
923         crate::vault::ErrorCode::UnbackedVaultBalance
924     );
925 }
```

Ordinary deposits therefore cannot redistribute the residual to a new holder or reopen subscriptions. Because the offset is denominated in raw units without regard to mint precision, either distortion can be economically material.

Exploit scenario

1. Alice, the last shareholder, owns 1,000,000 raw shares in Bob's vault after its NAV grows to 2,000,000 raw asset units.
2. Alice redeems all shares for 1,500,000 units with no exit fee and sufficient on-chain tokens, leaving 500,000 units of residual NAV while supply becomes zero.
3. Bob cannot reopen ordinary deposits while the residual remains.
4. Bob's vault alternatively holds 10,000,000 raw token units, owes 9,900,000 units in accrued fees, and has 10,000,000 outstanding shares backed by 100,000 units of NAV.
5. Alice redeems all shares for 1,000,000 units before any new exit fee, receiving 900,000 units more than the net backing and consuming tokens needed for the accrued fees.

Recommendation

Choose virtual asset and share units according to the mint precision and intended initial share price while retaining inflation resistance. Bound redemption, including the settlement fee, by real net backing. Reconcile residual assets when real share supply reaches zero so ordinary deposits can

resume without first-depositor capture.

Update 1.1

Full redemption now uses real net asset value, but partial conversions retain the virtual offset and can let a minority redemption consume all remaining backing.

Go back to Findings Summary

L21: Unset recipients can lock accrued fees

LOW SEVERITY ISSUE

Impact:	Low	**Likelihood:**	Medium
Type:	Data validation	**Status:**	Fixed
Target:	programs/fee_collection/src/lib.rs		

Description

Fee recipient initialization and `update_recipients` accept `Pubkey::default()`, allowing an accruing fee bucket to retain an unset recipient:

Listing 61. Excerpt from Unchecked recipient replacement

```
327 pub fn update_recipients(
328     ctx: Context<UpdateFeeConfig>,
329     recipients: FeeRecipients,
330 ) -> Result<()> {
331     ctx.accounts.vault_fees.recipients = recipients;
332     ctx.accounts.vault_fees.last_update = Clock::get()?.unix_timestamp;
333     Ok(())
334 }
```

A claim validates that the destination token account authority equals the stored recipient but does not reject the zero public key:

Listing 62. Excerpt from Destination authority check

```
693 #[account(
694     mut,
695     constraint = recipient_token_account.owner == recipient.key() @
    crate::vault::ErrorCode::InvalidRecipient
696 )]
697 pub recipient_token_account: Account<'info, TokenAccount>,
698
699 /// CHECK: recipient account
700 pub recipient: UncheckedAccount<'info>,
```

A compatible token account controlled by the zero address therefore passes validation. Because that address identifies the System Program and cannot sign an SPL token withdrawal, a successful claim makes the transferred fees inaccessible unless the account has a usable delegate.

A claim requires policy permission, a compatible initialized destination account, sufficient accrued fees and vault tokens, and enough recorded fee liability for the checked debit. Because `policy_limit_transfer` permits the `ReturnFunds` permission reused by claims, any authorized third party can trigger this loss under that policy; an administrator-gated policy limits the trigger to its administrator.

Exploit scenario

1. Bob, the fee administrator, leaves an accruing bucket's recipient at its default value.
2. Alice, a griefer authorized by a policy that permits `ReturnFunds`, claims the bucket into a compatible token account controlled by the zero address and without a delegate.
3. The tokens leave custody and become inaccessible; Bob's subsequent recipient update cannot recover them.

Recommendation

Reject `Pubkey::default()` for accruing recipients during initialization, recipient updates, and claims.

Represent intentionally disabled buckets with an explicit disabled state and prevent them from accruing fees.

Preserve the stored-recipient binding and all policy, liability, and custody checks.

Fix 1.1

A zero recipient now explicitly disables its fee bucket, disabled buckets cannot accrue, and an administrator can configure a valid recipient before accrual or claims.

Go back to Findings Summary

W1: Vault administrator can replace multisig membership alone

Impact:	Warning	**Likelihood:**	N/A
Type:	Trust model	**Status:**	Acknowledged
Target:	programs/policy_multisig/src/lib.rs, programs/policy_owners/src/lib.rs		

Description

`policy_multisig.update_owners` authorizes the live vault administrator to replace the owner set and threshold without approval from the outgoing owners:

Listing 63. Excerpt from administrator authorization for owner replacement

```
412 let vault_info = ctx.accounts.vault.to_account_info();
413 let vault = get_vault(&vault_info)?;
414 require_keys_eq!(vault.admin, ctx.accounts.admin.key(),
    ErrorCode::Unauthorized);
415
416 validate_owner_set(&owners, threshold)?;
```

A vault administrator can replace a two-of-three owner set with its own wallet, set the threshold to one, and clear pending approvals. The policy-program identifier remains unchanged even though the administrator's wallet becomes the only required approver. `policy_owners` uses the same administrator-only authorization model.

This authority permits recovery from an unusable owner set and is consistent with the administrator's existing control over transaction pausing and policy replacement. The multisig therefore constrains ordinary participants but does not protect membership or policy configuration from the administrator. Readers must inspect the policy state because the policy-program identifier

alone does not reveal the mutable membership or threshold.

Recommendation

Document the administrator's authority over membership and policy changes. Display the current membership, threshold, and recovery authority together.

For deployments that require protection from the administrator, require outgoing-owner authorization for membership and policy changes and define a separate delayed recovery process.

Acknowledgment 1.1

> `policy_multisig` is demo policy for documentation, out of audition scope
>
> — HyroTrader Team

Go back to Findings Summary

W2: Unauthenticated calls corrupt calculator fee state

Impact:	Warning	**Likelihood:**	N/A
Type:	Data validation	**Status:**	Fixed
Target:	programs/fee_collection_all_in_one/src/lib.rs, programs/fee_collection_time_based/src/lib.rs		

Description

Both stateful calculators use CPI depth as the only gate for `calculate_fees`. The configuration address is checked, but no signer authenticates the dispatcher because it calls the calculators with plain `invoke`.

Listing 64. Excerpt from CPI depth check without dispatcher authentication

```
// the returned fees. Require that we were reached by CPI from the fee dispatcher.
// Full caller authentication needs the dispatcher to pass a signing PDA (owned by
// `fee_collection`); this is the part enforceable from inside the calculator
// without changing the CPI account list.
require!(
    anchor_lang::solana_program::instruction::get_stack_height()
        > anchor_lang::solana_program::instruction::TRANSACTION_LEVEL_STACK_HEIGHT,
    ErrorCode::Unauthorized
);
```

Any program can therefore call `calculate_fees` against an existing configuration and modify accrual state. In `all_in_one`, a `Performance` call can raise the high-water mark using a fabricated balance. In both calculators, a `TimeBased` call can consume elapsed periods by advancing enabled buckets' timestamps. The time-only calculator has no high-water mark.

The core does not currently invoke `TimeBased` or `Performance` (I11), so these writes cannot directly credit the fee ledger. They can suppress fees after

those operations are integrated.

Recommendation

Have the dispatcher call both calculators with `invoke_signed` using a dedicated PDA, then add the PDA to both calculator interfaces and verify its address and signature instead of CPI depth. Until authentication is implemented, use the stateless fractional calculator, which has no accrual state to corrupt.

Fix 1.1

The dispatcher and calculators now require the canonical `vault_fees` account to be signed by a dedicated fee-authority PDA.

Go back to Findings Summary

W3: Closed challenge identities can be reused

Impact:	Warning	**Likelihood:**	N/A
Type:	Logic error	**Status:**	Reported
Target:	programs/policy_challenges/src/lib.rs		

Description

Administrator-authorized closure deletes a terminal challenge account, allowing the same child-address-derived PDA and identifier to be registered again as `Pending` without a persistent used-identity check:

Listing 65. Excerpt from Challenge account closure

```
312 #[account(
313     mut,
314     seeds = [challenge.vault.as_ref()],
315     bump = challenge.bump,
316     constraint = challenge.parent_vault == config.vault @
    ErrorCode::ChallengeVaultMismatch,
317     close = admin,
318 )]
```

The replacement can become active even though the original challenge reached a terminal state. Report validation checks only the current account's active status and parentage, while neither the incremented parent count nor changed timestamps form part of the challenge identity. Integrators and observers that identify a challenge only by its PDA or identifier can therefore mistake the replacement for the original terminal record unless they retain transaction history or compare creation timestamps.

Recommendation

Preserve a non-closable terminal record or persistent used-identity tombstone. If identity reuse is intentional, add a monotonic generation to the

externally consumed identity, distinguish reopening from creating a new generation, and require consumers to retain the generation when correlating qualification or payout records.

Update 1.1

Terminal challenges can no longer be deleted, but an active challenge can still return to `Pending`, close, and re-register the same child and challenge identifier.

Go back to Findings Summary

W4: Child allocation can leave the parent insolvent

Impact:	Warning	**Likelihood:**	N/A
Type:	Logic error	**Status:**	Fixed
Target:	programs/hyro_protocol/src/vault.rs		

Description

Funded child issuance checks an allocation only against the parent's actual and recorded on-chain balance, without deducting outstanding fee liabilities:

Listing 66. Excerpt from Allocation checks

```
require!(
    allocation <= parent_token_account.amount,
    ErrorCode::InsufficientFunds
);
require!(
    allocation <= ctx.accounts.parent_vault.onchain_balance,
    ErrorCode::InsufficientFunds
);
```

The transfer removes the allocated tokens from the parent's on-chain balance. Because the parent NAV is `onchain + offchain - obtained_fees`, an allocation that exceeds net NAV makes the parent insolvent and causes subsequent deposit and withdrawal settlements to fail. This operation requires the matching parent and registry administrator, a verified manager, and parent-policy approval.

Recommendation

Validate that the complete post-allocation NAV remains nonnegative before transferring funds. Reserve sufficient on-chain liquidity if fees must remain immediately claimable. Define whether child capital remains a parent-owned asset and, if so, record a child receivable in the parent's NAV.

Fix 1.1

Funded child allocation now checks the parent token balance, recorded on-chain balance, and total balance before transferring and recording the allocation.

Go back to Findings Summary

W5: CI omits account-layout regression checks

Impact:	Warning	**Likelihood:**	N/A
Type:	Logic error	**Status:**	Fixed
Target:	`.github/workflows/ci.yml`, `programs/fee_collection/src/lib.rs`, `lib/hyro_sdk/src/migrate.rs`		

Description

CI builds and uploads protocol IDLs without running the protocol tests, and the publishing workflow repeats this build-only process (`.github/workflows/ci.yml:17-54`). Consequently, CI omits the four fee programs' current and historical account-layout size assertions and the SDK migration-error-range check because they are gated by `#[cfg(test)]`.

Current allocation lengths derive from `INIT_SPACE` instead of frozen literals. For example, adding a `u64` to `VaultFees` increases its computed allocation from 345 to 353 bytes while the build still succeeds, although the existing 345-byte layout assertion would fail. CI therefore does not require review of historical layouts and migration support when account layouts change.

Recommendation

Run the existing account-layout and migration unit tests as required CI checks alongside IDL generation. Include protocol integration tests in release validation. Review historical lengths and migration behavior whenever layout pins or their expected values change.

Fix 1.1

Continuous integration now runs `cargo test --workspace`, which includes the retained account-layout and error-range tests.

Go back to Findings Summary

W6: Fee claims inherit capital-return permissions

Impact:	Warning	**Likelihood:**	N/A
Type:	Access control	**Status:**	Fixed
Target:	programs/hyro_protocol/src/funds.rs		

Description

`claim_fees` authorizes fee payments as `ReturnFunds`, so any policy that permits capital returns also permits callers to decide when accrued fees are paid to the configured recipient:

Listing 67. Excerpt from Fee claims use capital-return authorization

```
1776     &vec![
1777         ctx.accounts.policy_account.to_account_info(),
1778         ctx.accounts.signer.to_account_info(),
1779     ],
1780     ValidateOperation::ReturnFunds,
1781 )?;
```

`ClaimFees` also omits the Instructions sysvar. The SDK accepts this two-account context and returns `None`, preventing policies from inspecting sibling Ed25519 verification instructions. Policies that require this context fail closed, but permissive `ReturnFunds` policies such as `policy_limit_transfer` allow fee payments without distinct fee-claim permission.

Recommendation

Add a dedicated fee-claim operation to both sides of the ABI and explicitly authorize it in every policy. Add an address-checked Instructions sysvar to `ClaimFees` and forward it to the policy. Preserve the recipient and balance checks, and fail closed when required verification context is absent.

Fix 1.1

Fee claims now use a dedicated `ClaimFees` policy operation instead of reusing `ReturnFunds` authorization.

Go back to Findings Summary

W7: Fee dispatch receives custody authority

Impact:	Warning	**Likelihood:**	N/A
Type:	Access control	**Status:**	Fixed
Target:	programs/hyro_protocol/src/helpers.rs		

Description

Fee dispatch uses `share_signer_pda`, which controls the share mint and primary token account, as the misleadingly named `vault_authority`:

Listing 68. Excerpt from Fee dispatch receives the share signer PDA

```
63 let bump = [share_signer_bump];
64 let seeds: &[&[u8]] = &[b"vault_share_signer", vault_key.as_ref(), &bump];
65 let signer = &[seeds];
66
67 solana_program::program::invoke_signed(&instruction, &account_infos,
   signer)?;
```

All five charging callers pass this PDA instead of the separate proposed-execution authority, and the dispatcher verifies the same seeds. A dispatcher can therefore exercise custody authority when caller-selected fee slots contain the vault's writable token account, a writable same-mint recipient account, and a compatible token program.

Operation-policy authorization precedes each fee call, so a rejected operation cannot reach dispatch. The dispatcher is selected at vault creation, inherited by child vaults, and cannot be replaced through this path. However, dispatcher upgrade governance remains a custody trust boundary because a malicious upgrade can transfer custody tokens during any policy-authorized operation that supplies the required accounts.

Recommendation

Authenticate fee dispatch with a dedicated core-derived PDA that has no token-account or mint authority, and migrate the dispatcher verifier and all charging callers, including claims, together. Rename parameters to identify the actual signer and retain operation-policy checks. Explicitly track dispatcher governance as a custody trust boundary until the excess authority is removed.

Fix 1.1

Fee collection CPIs now use a custody-less fee-authority PDA instead of the vault share signer.

Go back to Findings Summary

W8: Timestamp resets discard unsettled fee periods

Impact:	Warning	**Likelihood:**	N/A
Type:	Logic error	**Status:**	Reported
Target:	programs/fee_collection_all_in_one/src/lib.rs, programs/fee_collection_time_based/src/lib.rs		

Description

The stored configuration authority can reset all four fee timestamps without settling elapsed fee periods:

Listing 69. Excerpt from Timestamp reset without fee settlement

```
429 let now = Clock::get()?.unix_timestamp;
430 let config_account = &mut ctx.accounts.config_account;
431 config_account.last_lp_calc = now;
432 config_account.last_manager_calc = now;
433 config_account.last_protocol_calc = now;
434 config_account.last_performance_calc = now;
```

The next calculation begins at the reset time and omits fees for the discarded interval from the ledger. Changes to the rate, mode, or period also lack checkpointing: a subsequent reset discards the interval under the old terms, while a change without a reset applies the new terms from the old timestamp. No application event identifies the discarded interval, although the transaction history records the reset.

The core does not emit `TimeBased`, so these intervals are not currently collectible protocol liabilities and neither elapsed time nor configuration changes accrue fees autonomously. The stored authority may remain authorized after administrator rotation (H2).

Recommendation

Define an explicit authorization mechanism for fee waivers and emit the old and new timestamps for each waiver. Before enabling time-based fee settlement, checkpoint elapsed fees under the old terms before changing the rate, mode, period, or timestamps. Provide a separately authorized recovery path for unpayable backlogs.

Update 1.1

Reset and configuration-change events were added, but resets still discard unpaid periods and configuration updates can reprice already elapsed periods.

Go back to Findings Summary

W9: Performance calculations misclassify principal as profit

Impact:	Warning	**Likelihood:**	N/A
Type:	Logic error	**Status:**	Reported
Target:	programs/fee_collection_all_in_one/src/lib.rs, programs/fee_collection_fractions/src/lib.rs		

Description

Both performance-fee calculators treat any balance above the high-water mark as profit without adjusting the mark for new capital:

Listing 70. Excerpt from Performance fee calculation above the high-water mark

```
550 if current_balance > config_account.high_water_mark {
551     let profit = current_balance
552         .checked_sub(config_account.high_water_mark)
553         .ok_or_else(|| error!(ErrorCode::MathOverflow))?;
554     let performance_fee = bps_of(profit,
    config.fraction.performance_fee_bps)?;
```

This behavior becomes active if a future integration invokes `Performance` with a nonzero fee rate; the current core does not invoke it. It can charge performance fees on principal in share-issuing vaults that accept deposits and child vaults that receive additional capital without a corresponding mark adjustment. A fixed-capital child with a correctly initialized and maintained mark is unaffected, and a fee calculation for a virtual child alone does not establish a real-token loss.

Initialization can independently misclassify starting capital. All-in-one initializes its separate mark to zero and ignores the forwarded mark, while Fractions uses the forwarded mark. Child issuance seeds the core mark to the allocation, but root initialization leaves it at zero. The first applicable calculation therefore treats starting capital as profit unless the relevant mark

is seeded.

Authorized mark updates must also remain consistent: All-in-one permits downward resets, whereas the dispatcher does not.

Recommendation

Initialize every performance-fee mark from the applicable starting capital, including the All-in-one mark and root initialization.

Adjust each mark for deposits and withdrawals or calculate gains per share, while preserving correctly seeded child allocations.

Define consistent authorization and behavior for recovery resets across the calculators and dispatcher.

Update 1.1

High-water marks remain inconsistent, and the all-in-one initializer writes the current net asset value before immediately overwriting the mark with zero.

Go back to Findings Summary

W10: Child deposits can open without a usable share mint

Impact:	Warning	**Likelihood:**	N/A
Type:	Logic error	**Status:**	Reported
Target:	programs/hyro_protocol/src/vault.rs, programs/hyro_protocol/src/funds.rs		

Description

`issue_child_vault` does not create or record the canonical share mint, and no instruction completes this initialization:

Listing 71. Excerpt from Missing child share-mint initialization

```
1021 child_vault.admin = ctx.accounts.admin.key();
1022 child_vault.share_mint = Pubkey::default(); // set when the child is
     fully initialized
1023 child_vault.underlying_mint = ctx.accounts.parent_vault.underlying_mint;
     // Inherit from parent
1024 child_vault.manager = Some(ctx.accounts.manager.key());
1025 child_vault.parent_vault = Some(ctx.accounts.parent_vault.key());
```

An administrator can enable deposits before the share mint exists. If the child's policy admits a deposit, the protocol escrows the underlying assets, but settlement cannot issue shares because it requires both the canonical share-mint PDA and `vault.share_mint`. `policy_challenges` rejects deposits, so enabling deposits alone is insufficient to reach this state. The owner can cancel the request and recover the escrowed assets without policy approval or a share mint.

Child vaults also require per-vault fee ledgers and calculator configurations for fee-dependent operations. Assigning or inheriting `fee_collection_program` does not create them, reporting requires three fee accounts and invokes charging, and no parent-ledger fallback exists. Existing administrator-gated

fee initializers can provision these accounts, but no equivalent share-mint completion flow exists.

Recommendation

Create and record the canonical share mint through an authenticated child-completion flow before enabling deposits or redemptions, or explicitly reject unsupported child subscriptions. Preserve cancellation for requests that cannot settle. Provision each child's fee ledgers and calculator configurations before enabling fee-dependent operations, including for virtual children. Correct the issuance comments and documentation to describe funded-child token-account creation and allocation transfers, virtual-child allocation behavior, and the separate share-mint and fee-account prerequisites.

Update 1.1

Funded-child completion creates a share mint but attributes no shares to the existing allocation, leaving positive net asset value with zero share supply.

Go back to Findings Summary

W11: Manager permissions grant full staged-fund custody

Impact:	Warning	**Likelihood:**	N/A
Type:	Trust model	**Status:**	Acknowledged
Target:	programs/policy_manager_access/src/lib.rs, programs/hyro_protocol/src/funds.rs, programs/hyro_protocol/src/vault.rs		

Description

`policy_manager_access` applies the same administrator-or-manager authorization to `Creation`, `Execution`, and `UseFunds`; the manager branch checks only that the signer is registered:

Listing 72. Excerpt from Registered-manager authorization

```
289 let policy = load_managers(policy_account)?;
290 require!(
291     policy.managers.contains(signer),
292     ErrorCode::UnregisteredManager
293 );
294 Ok(())
```

After policy approval, `use_funds` moves capital from the share-signer-authorized token account into a vault-authority-authorized token account. An approved arbitrary CPI then receives the vault-authority signature and can transfer the staged capital to a manager-chosen destination, although the separate share signer remains unavailable to the CPI. An administrator-appointed manager therefore has custody of staged capital rather than constrained trading authority.

This authority contradicts the documented noncustodial boundary and the claimed equivalence between `policy_manager_access` and the retired policy, which rejects every validation:

Listing 73. Excerpt from Retired policy rejects all operations

```
55 pub fn validate<'a, 'b, 'c: 'info, 'info>(
56     _ctx: Context<'a, 'b, 'c, 'info, Validate<'info>>,
57     _operation: ValidateOperation,
58 ) -> Result<()> {
59     // Fails closed: authorizes nothing, for any operation, ever.
60     Err(error!(ErrorCode::Deprecated))
61 }
```

Recommendation

Choose and enforce one manager authority model:

- For constrained delegation, separate staging permission from arbitrary proposal creation and execution, and restrict permitted programs, accounts, and destinations.
- For custody, disclose that managers can transfer staged capital and narrow the noncustodial claims.

Correct the retired-policy equivalence claim and document the authority granted by each selectable policy.

Acknowledgment 1.1

> `policy_manager_access` is demo policy for documentation, out of audition scope
>
> — HyroTrader Team

Go back to Findings Summary

W12: Missing signature context blocks approval-gated vault operations

Impact:	Warning	**Likelihood:**	N/A
Type:	Logic error	**Status:**	Fixed
Target:	lib/hyro_sdk/src/validate.rs, programs/hyro_protocol/src/vault.rs, programs/hyro_protocol/src/funds.rs		

Description

Child issuance and fee claims omit the Instructions sysvar from their policy calls, so policies cannot inspect a sibling Ed25519 approval. Child issuance forwards four accounts and provides no SDK sysvar accessor:

Listing 74. Excerpt from child issuance policy account layout

```
1002 &[
1003     ctx.accounts.parent_policy_account.clone(),
1004     ctx.accounts.admin.to_account_info(),
1005     ctx.accounts.child_vault.to_account_info(),
1006     ctx.accounts.manager.to_account_info(),
1007 ],
```

Fee claims reuse `ReturnFunds`, whose context expects the sysvar at index 2, but forward only two accounts:

Listing 75. Excerpt from fee claim policy account layout

```
1776 &vec![
1777     ctx.accounts.policy_account.to_account_info(),
1778     ctx.accounts.signer.to_account_info(),
1779 ],
1780 ValidateOperation::ReturnFunds,
```

A custom policy that requires a sibling Ed25519 approval for either operation

must reject it even when the administrator includes a valid verification instruction and the Instructions sysvar in the outer remaining accounts. Existing policies do not require this approval for these operations, and policies can consume previously stored approvals.

Recommendation

Add a canonically constrained Instructions sysvar to both core account contexts and forward it at child index 4 and claim index 2. Add the child SDK accessor, and update the instruction builders and both interfaces because optional accessors do not make the new outer accounts backward-compatible. Require signature-dependent policies to fail closed and bind approvals to the operation. Give fee claims a distinct tag and context as specified in W6, appending the tag without changing existing discriminants.

Fix 1.1

Child issuance and fee claims now receive and forward the canonical Instructions sysvar through distinct policy contexts. The out-of-scope generated Rust client was not updated for the new accounts.

Go back to Findings Summary

W13: Policy helpers leave ownership checks to callers

Impact:	Warning	**Likelihood:**	N/A
Type:	Data validation	**Status:**	Reported
Target:	lib/hyro_sdk/src/validate.rs		

Description

SDK request accessors return raw accounts without enforcing the transaction-level ownership guarantee (`lib/hyro_sdk/src/validate.rs:574-579`, `lib/hyro_sdk/src/validate.rs:703-708`). `policy_checkout` deserializes a deposit request by discriminator without verifying that the account is owned by the expected program:

Listing 76. Excerpt from Discriminator-checked request deserialization

```
// Deserialize the DepositRequest (Anchor account -> discriminator-
checked).
let dr_account = vctx.deposit_request(remaining);
let dr = {
    let data = dr_account.try_borrow_data()?;
    DepositRequest::try_deserialize(&mut &data[..])?
};
```

The core creates the typed request, transfers the deposit, stores its fields, and passes the same account to checkout. Consequently, the missing ownership check does not currently permit a forged-request fund bypass because direct checkout validation moves no core funds, creates no reusable authorization, and requires a matching administrator-signed voucher. However, callers can mistake discriminator-checked request data for authenticated request state.

The SDK also accepts either the canonical policy-state PDA or the policy program ID without checking ownership or initialization:

Listing 77. Excerpt from Policy-account address check

```
188     let (expected_pda, _) =
    Pubkey::find_program_address(&[vault.as_ref()], policy_program_id);
189     require!(
190         policy_account.key() == expected_pda || policy_account.key() ==
    *policy_program_id,
191         ErrorCode::InvalidPolicyAccount
192     );
193     Ok(())
194 }
```

A stateful policy can therefore receive its executable program account as state, and an uninitialized policy-state PDA also passes this helper. Current policies prevent an established authorization bypass through additional checks: manager and challenge policies verify ownership, challenge validation binds the address and vault, and `policy_limit_transfer` requires a discriminator. For example, `policy_manager_access` rejects its loader-owned executable account despite the SDK address check accepting it.

These helpers therefore do not establish that request or policy-state accounts are initialized and owned by the expected program, leaving these authentication boundaries to each caller. The architecture reference incorrectly claims that the core enforces policy-account ownership.

Recommendation

Provide owner-checked request readers and use them wherever request fields authorize actions. Require policy-program ownership and initialization for state PDAs, and validate stateless policy markers through a separate explicit path. Document each accessor's guarantees and caller obligations, correct the obsolete architecture claim, and retain all existing discriminator, address, vault-binding, and signer checks.

Update 1.1

Policy-state helpers now verify address and owner, but they still accept

program-ID markers without proving executable stateless behavior or initialized state.

Go back to Findings Summary

W14: Authorized proposals have no enforced response window

Impact:	Warning	**Likelihood:**	N/A
Type:	Logic error	**Status:**	Acknowledged
Target:	programs/hyro_protocol/src/vault.rs		

Description

The core records no proposal timestamp or hold-up interval. Proposal execution obtains a fresh policy verdict and immediately invokes the stored CPI with the vault-authority signature, so a policy-authorized principal can create and execute a proposal in one outer transaction (programs/hyro_protocol/src/vault.rs:764-894). A monitor cannot cancel, pause, or otherwise respond between those instructions.

This missing response window affects assets and permissions reachable through the vault-authority PDA, but not the separate share signer's primary balance. Execution still requires policy authorization, and a stateful policy can enforce its own timing rule.

Recommendation

Define and enforce a hold-up interval for sensitive arbitrary CPIs in the core or an explicitly required policy. Record the proposal's creation or authorization time and reject execution before the interval expires, and prevent delay reductions from bypassing safeguards for pending proposals. Provide cancellation or pause controls to the responsible authority. Disclose configurations that intentionally permit immediate execution.

Acknowledgment 1.1

> Design of the system to support synchronous operations (such

> as trading on decentralized dexes)
>
> — HyroTrader Team

Go back to Findings Summary

W15: Proposal approval does not bind target program code

Impact:	Warning	**Likelihood:**	N/A
Type:	Trust model	**Status:**	Acknowledged
Target:	programs/hyro_protocol/src/vault.rs		

Description

Proposal creation stores a target address and payload without binding approval to the target's executable code version:

Listing 78. Excerpt from proposal target and payload storage

```
793 tx.nonce = nonce;
794 tx.vault = ctx.accounts.vault.key();
795 tx.program_id = pid;
796 tx.data = data;
797 tx.accounts = accs;
```

Execution invokes the stored target with the vault-authority PDA signature. Because the target can be deployed after proposal creation or upgraded before execution, the stored address and payload can invoke code that approvers did not review. Configured policy approval and a valid executable target remain required, and direct core calls remain rejected, but approvers must trust the target's deployment and upgrade authorities until execution. An address allowlist does not bind the approved implementation.

For example, approvers review payload `D` for version 1 of upgradeable program `P`. The upgrade authority installs version 2 before execution, so the approved address and payload invoke version 2 if the execution-policy check approves the proposal.

Recommendation

Include target upgradeability in approval decisions. Restrict sensitive proposals to trusted immutable targets or record the reviewed code version and verify it at execution. Invalidate and reapprove proposals after implementation changes. Document separately which controls bind target addresses and which bind code versions.

Go back to Findings Summary

W16: Request handlers omit core policy-state binding

Impact:	Warning	**Likelihood:**	N/A
Type:	Data validation	**Status:**	Fixed
Target:	programs/hyro_protocol/src/funds.rs		

Description

Eight deposit and withdrawal request, settlement, approval, and rejection handlers forward caller-supplied policy state without applying the core policy-state address binding used by fund-management handlers.

All four approval and rejection handlers invoke policy validation directly, and the writable policy-account fields in all four request and settlement contexts have only a mutable constraint despite comments stating that the core validates their addresses:

Listing 79. Excerpt from Request policy-account field

```
53 /// CHECK: policy account - will be validated for correct PDA derivation
   (mutable for clearing pending transaction)
54 #[account(mut)]
55 policy_account: AccountInfo<'info>,
```

A custom policy that verifies only program ownership and the signer may rely on the core to bind its state to the target vault. Such a policy can receive valid state associated with another vault because these handlers omit that binding, creating inconsistent integration guarantees between request handling and fund management.

The existing `policy_owners`, `policy_manager_access`, and `policy_multisig` policies bind their own state, withdrawal-window settlement also applies a binding, and pass-through policies do not use substitute accounts. The gap therefore does not currently establish an authorization bypass, but it can

expose custom policies that rely on the documented core check.

Recommendation

Apply the same core policy-state address binding before CPI in all eight handlers. Retain policy-local state binding as defense in depth. Update the policy-account field comments to identify the layer that enforces the binding.

Fix 1.1

All request, settlement, approval, and rejection handlers now validate the canonical policy account before invoking the policy program.

Go back to Findings Summary

W17: Policy replacement preserves pending proposals

Impact:	Warning	**Likelihood:**	N/A
Type:	Logic error	**Status:**	Acknowledged
Target:	programs/hyro_protocol/src/vault.rs		

Description

`update_vault_policy` replaces a paused vault's policy pointer without invalidating its pending `Transaction` accounts, which contain no policy identifier or epoch:

Listing 80. Excerpt from Policy replacement

```
1303 let config =
     vault.config.as_ref().ok_or(ErrorCode::PolicySwapRequiresTransactionsDisa
     bled)?;
1304 require!(
1305     config.transactions_disabled,
1306     ErrorCode::PolicySwapRequiresTransactionsDisabled
1307 );
1308
1309 vault.policy_program = new_policy_program;
1310 Ok(())
```

After the vault resumes, each surviving proposal is validated against the current policy. A proposal that the new policy rejects cannot execute, while one that it accepts remains executable. Policy replacement therefore does not discard the pending queue, contrary to an operator's possible expectation, but it does not bypass the current policy. The included multisig, transfer-limit, owner, and manager policies respectively check pending state, amount at execution, or the current executor.

For example, Bob, a vault operator, pauses the vault and replaces its policy expecting to retire all pending proposals. Alice's earlier proposal survives and can execute after the vault resumes if the new policy approves it.

Recommendation

Define whether policy replacement preserves or invalidates pending proposals. If it invalidates them, add a policy epoch to the vault and each proposal and provide authenticated cancellation. Otherwise, document revalidation under the current policy and expose surviving proposals to operators.

Go back to Findings Summary

W18: Time-fee calculations misprice unsettled periods

Impact:	Warning	**Likelihood:**	N/A
Type:	Logic error	**Status:**	Acknowledged
Target:	programs/fee_collection_all_in_one/src/lib.rs, programs/fee_collection_time_based/src/lib.rs		

Description

Both stateful time-fee calculators apply the current `input.total_balance` to every unsettled period and floor one period's fee before multiplying it by the number of periods:

Listing 81. Excerpt from Time-based fee calculation for unsettled periods

```
745 let fee_per_period = (total_balance as u128)
746     .checked_mul(entry.value as u128)
747     .ok_or_else(|| error!(ErrorCode::MathOverflow))?
748     .checked_div(10_000)
749     .ok_or_else(|| error!(ErrorCode::MathOverflow))?;
750 let total = fee_per_period
751     .checked_mul(periods as u128)
752     .ok_or_else(|| error!(ErrorCode::MathOverflow))?;
```

If the balance changes between periods, backlog settlement overcharges or undercharges because historical balances are not checkpointed. Per-period flooring can also lose almost one base unit for each unsettled period instead of rounding the aggregate once.

Successful settlement advances the calculator timestamp. The 10,000-basis-point entry and summed-rate ceilings apply per period rather than to the backlog. The dispatcher accepts a fee equal to the positive vault balance and atomically rejects a larger fee, including the timestamp update, instead of truncating collection (`programs/fee_collection/src/lib.rs:412-428`). A sufficiently large backlog can therefore prevent settlement and leave accrual

progress blocked.

Core accounting's five fee calls currently emit neither `TimeBased` nor `Performance`, so the defects remain dormant until an integration enables time-based accrual.

Recommendation

Checkpoint historical or time-weighted assets before capital changes and calculate each unsettled period from its applicable balance. Multiply by the period count before division where a single balance applies, using checked wide arithmetic and converting only after the final division. Bound backlog settlement and define explicit accrual and recovery behavior when accrued fees exceed available assets.

Acknowledgment 1.1

> `fee_collection_all_in_one and fee_collection_time_based are out of audition scope`
>
> `— HyroTrader Team`

Go back to Findings Summary

W19: Account growth can preserve stale field values

Impact:	Warning	**Likelihood:**	N/A
Type:	Data validation	**Status:**	Acknowledged
Target:	lib/hyro_sdk/src/migrate.rs		

Description

`grow_account<T: Discriminator>` verifies ownership, the discriminator, and allocated lengths, but it neither finds the end of the old serialized value nor rewrites the value. Its `resize` call zeroes only newly allocated bytes:

Listing 82. Excerpt from Allocation-tail zeroing

```
190 // `resize` is `realloc(new_len, zero_init = true)`. The zero-fill is load
191 // bearing, not incidental: it is what makes the appended fields decode as
192 // their zero values, and it stops stale heap contents leaking into the
    tail.
193 account.resize(target_len)?;
```

In a variable-length Borsh layout, appended fields begin after the actual serialized value and can occupy existing allocation slack. `store_account` serializes a value without clearing that slack:

Listing 83. Excerpt from Account write without slack clearing

```
216 pub fn store_account<T: AccountSerialize>(account: &AccountInfo, value:
    &T) -> Result<()> {
217     let mut data = account.try_borrow_mut_data()?;
218     let mut cursor: &mut [u8] = &mut data;
219     value.try_serialize(&mut cursor)?;
220     Ok(())
221 }
```

Shortening a value can therefore leave bytes that a later appended field decodes as its default value. The module overview directs variable-length

layouts to a bespoke decoder, but the helper's generic signature does not enforce this restriction. Future automatic migrations of `Vault` or `Managers` are susceptible because their layouts contain strings, optional fields, or vectors; changing `max_capacity` from `Some` to `None` is one operation that shortens serialization.

The four current migration targets, `VaultFees`, `VaultFractionConfig`, `VaultTimeBasedConfig`, and `VaultAllInOneConfig`, have fixed layouts and do not trigger this behavior.

Recommendation

Restrict automatic zero defaults to layouts whose actual serialization length equals the expected legacy length. For variable-length layouts, explicitly select and deserialize the correct legacy schema, initialize appended fields, and serialize the complete new value after growth. Clear unused account bytes when storing shortened values.

Go back to Findings Summary

I1: Reporting equity before bootstrap blocks the first deposit

Impact:	Info	**Likelihood:**	N/A
Type:	Logic error	**Status:**	Reported
Target:	programs/hyro_protocol/src/funds.rs		

Description

`deposit` rejects the first deposit when a zero-share vault has nonzero net asset value:

Listing 84. Excerpt from Deposit bootstrap guard

```
920 if ctx.accounts.share_mint.supply == 0 {
921     require!(
922         ctx.accounts.vault.total_balance == 0,
923         crate::vault::ErrorCode::UnbackedVaultBalance
924     );
925 }
```

An authorized oracle can report equity and incur report fees before bootstrap, leaving positive net asset value without corresponding shares. The bootstrap guard then prevents the first depositor from capturing that value by returning `UnbackedVaultBalance`; negative net asset value instead triggers the earlier insolvency check. The vault therefore cannot accept its first deposit until the report or initialization state is corrected.

Recommendation

Reject equity reports before bootstrap unless the vault supports prefunding.

For prefunded vaults, add an administrator-controlled initialization flow that truthfully attributes equity and issues corresponding shares before ordinary deposits.

Document how authorized roles correct erroneous reports and settle fee liabilities without misreporting asset values.

Update 1.1

The report bootstrap guard checks whether both recorded balances are zero instead of checking share supply, so it blocks legitimate recoveries after a complete loss and permits some zero-supply states.

Go back to Findings Summary

I2: Parent reports fail with an unrelated account-type error

Impact:	Info	**Likelihood:**	N/A
Type:	Logic error	**Status:**	Acknowledged
Target:	programs/policy_challenges/src/policy.rs		

Description

The challenge policy loads and validates a typed `Challenge` before checking its parentage:

Listing 85. Excerpt from Challenge loading before parentage validation

```
let challenge = load_challenge(report_ctx.policy_account(remaining),
&vault_key)?;
require!(
    challenge.status == ChallengeStatus::Active,
    ErrorCode::ChallengeNotActive
);
```

A parent's canonical policy account contains `ChallengeConfig`, so an authorized oracle operator who selects the parent and supplies that account receives a deserialization error instead of `NotAChildVault`. The operation remains rejected without an unauthorized equity update. `NotAChildVault` remains reachable for a valid `Challenge` with inconsistent parentage.

Recommendation

Check `parent_vault == None` and return `NotAChildVault` before loading the challenge. Then retain the registered-parent equality check and the canonical-PDA, owner, and discriminator checks.

Go back to Findings Summary

I3: Documented challenge seeds exceed the PDA limit

Impact:	Info	**Likelihood:**	N/A
Type:	Data validation	**Status:**	Fixed
Target:	programs/hyro_protocol/src/vault.rs		

Description

The documented `ch-<uuid>` child-vault seed convention can exceed Solana's 32-byte per-seed limit because the program accepts the seed as a UTF-8 `String`. Prefixing a 36-character UUID produces a 39-byte seed, so address derivation fails before the handler's length guard and no child-vault address is produced.

Listing 86. Excerpt from Documented child-vault seed

```
554 // New child vault. Addressed like a root vault: [parent, seed]. The
    caller
555 // picks a `seed` that is unique under the parent (ht-core uses "ch-
    <id>").
```

Registration also documents `[parent, seed, allocation, manager]`, while the account constraint derives the address from only `[parent, seed]`. An integrator that registers an independently supplied child address before derivation can create an unmatched row, although an administrator can mark it terminal and close it.

Listing 87. Excerpt from Actual child-vault seeds

```
556 #[account(
557     init,
558     payer = admin,
559     seeds = [parent_vault.key().as_ref(), seed.as_ref()],
560     bump,
561     space = 8 + Vault::INIT_SPACE
562 )]
```

```
563 pub child_vault: Box<Account<'info, Vault>>,
```

Recommendation

Define a canonical UTF-8 seed encoding that remains within 32 bytes, such as a compact textual UUID with the prefix. Change the `String` interface if seeds must contain arbitrary raw UUID bytes. Require derivation and validation before separate registration. Correct the registration documentation to specify `[parent, seed]`.

Fix 1.1

The child-vault seed convention is now documented consistently, and challenge registration verifies the child PDA against its parent and numeric seed.

Go back to Findings Summary

I4: Checkout documentation overstates voucher binding

Impact:	Info	**Likelihood:**	N/A
Type:	Logic error	**Status:**	Reported
Target:	programs/policy_checkout/src/lib.rs, programs/policy_checkout/src/message.rs		

Description

The checkout documentation states that vouchers bind the payer and mint:

Listing 88. Excerpt from Checkout binding claims

```
10 //! 2. covers a [`CheckoutMessage`] (`CHECKOUT_MSG_V2`) whose `vault`,
11 //!    `depositor`, `underlying_mint`, `amount` and `memo` exactly match
   this
12 //!    deposit's `DepositRequest`, and whose `expires_at` has not passed.
13 //!
14 //! AUDIT H-07: `CHECKOUT_MSG_V1` bound only `(amount, memo)`, which made
   every
15 //! on-chain checkout signature a reusable bearer token (any vault sharing
   the
16 //! admin, any depositor, forever). v2 binds the vault, the depositor, the
   mint
17 //! and an expiry — see [`message`] for the byte layout and the ht-core
   contract.
```

However, the signed message contains only the versioned domain, vault, amount, expiry, and length-prefixed memo:

Listing 89. Excerpt from Encoded voucher fields

```
105 let mut out = Vec::with_capacity(MEMO_OFFSET + self.memo.len());
106 out.extend_from_slice(&CHECKOUT_DOMAIN);
107 out.extend_from_slice(self.vault.as_ref());
108 out.extend_from_slice(&self.amount.to_le_bytes());
109 out.extend_from_slice(&self.expires_at.to_le_bytes());
110 out.extend_from_slice(&memo_len.to_le_bytes());
111 out.extend_from_slice(self.memo.as_bytes());
```

The vault determines the mint, but the message does not bind a payer. Validation requires the request owner to sign without requiring that owner to match a signed payer. A voucher therefore authorizes any holder to deposit the specified amount into the vault before expiry and remains reusable on-chain. Whether reuse grants duplicate benefits depends on the backend's order association, deduplication, and entitlement checks.

The implementation deliberately uses bearer authorization, but the contradictory documentation can cause integrators to rely on payer binding or on-chain single-use enforcement that does not exist.

Recommendation

Correct the checkout header and prior-fix narrative to document the actual voucher fields, bearer authorization, vault-derived mint binding, and absence of on-chain single-use enforcement.

If payer restriction or single use is required, sign and validate the payer, and add a signed nonce with consumed state. Define settlement expiry separately from request expiry.

Verify whether multiple deposits with the same order identifier can grant multiple challenges or other entitlements, and document the backend check that enforces one-time fulfillment.

Update 1.1

Most bearer-voucher documentation was corrected, but `message.rs` still describes the message as depositor-, mint-, and expiry-bound, and the claimed external atomic fulfillment was not supplied.

Go back to Findings Summary

I5: Core account growth lacks explicit layout safeguards

Impact:	Info	**Likelihood:**	N/A
Type:	Code quality	**Status:**	Fixed
Target:	programs/hyro_protocol/src/vault.rs, programs/hyro_protocol/src/manager_registry.rs		

Description

`ManagerRegistry`, `ManagerProfile`, and `VaultOracle` allocate exactly their current serialized sizes of 60, 78, and 73 bytes, including discriminators, without reserved extension fields, layout-version tags, or migration safeguards.

`Transaction` likewise reserves only its serialized header and variable payload:

Listing 90. Excerpt from Transaction allocation size

```
69 pub const fn transaction_space(num_accounts: usize, data_len: usize) ->
   usize {
70     // 8 discriminator
71     // + 8 nonce + 1 did_execute + 32 vault + 32 program_id
72     // + 4 data len prefix + data
73     // + 4 accounts len prefix + accounts
74     8 + 8 + 1 + 32 + 32 + 4 + data_len + 4 + num_accounts *
   TRANSACTION_ACCOUNT_SIZE
75 }
```

Its allocation is `89 + data.len() + 34 * accounts.len()` bytes, rather than a fixed 89-byte account.

`Vault`, `DepositRequest`, and `WithdrawalRequest` reserve optional or variable-length maxima, so their unused capacity depends on stored values, and a fully populated `Vault` consumes all 450 bytes.

Changing `INIT_SPACE` does not grow existing accounts, while incidental zero-

filled space neither guarantees extension capacity nor defines new-field semantics.

Appending fields can therefore make existing core accounts incompatible with upgraded deserialization and cause otherwise valid operations to fail.

Recommendation

Assert serialized-size and allocation contracts for maximum-sized values and `Transaction` payloads.

Implement source-version-specific migrations for incompatible layout changes that validate legacy bytes before decoding, explicitly resize existing accounts when increasing `INIT_SPACE`, and initialize newly introduced fields with defined semantics.

Fix 1.1

New accounts use versioned layouts, reserved extension space, and pinned size tests, which provide explicit layout safeguards for future deployments. Existing vaults are not safely migrated because the migration decodes the new layout from uncleared legacy slack.

Go back to Findings Summary

I6: Dead modules misrepresent supported core features

Impact:	Info	**Likelihood:**	N/A
Type:	Code quality	**Status:**	Fixed
Target:	programs/hyro_protocol/src/lib.rs, programs/hyro_protocol/Cargo.toml		

Description

The core source advertises modules and feature-gated entrypoints that no supported build exposes. `lib.rs` does not declare `multisig.rs`, which contains a separate `coral_multisig` program, or the empty `states.rs`, so neither defines a live core instruction or security boundary.

The governance, oracle, and verification entrypoints depend on `enable_governance`, `enable_oracle`, and `enable_verification`, but `Cargo.toml` declares none of these features:

Listing 91. Excerpt from Declared core features

```
[features]
default = []
cpi = ["no-entrypoint"]
no-entrypoint = []
no-log-ix-name = []
idl-build = ["anchor-lang/idl-build", "anchor-spl/idl-build"]
mainnet = []
```

Defining the gates alone would not make the entrypoints usable because they reference unresolved `Initialize*` contexts and `initialize_*` handlers, while their modules contain only `Placeholder*` contexts and `placeholder_*_instruction` functions. The repository therefore misrepresents unsupported core integration paths, such as `initialize_oracle`, as available features even though the live `set_vault_oracle` instruction is separate.

Recommendation

Remove unused modules, placeholders, and unsupported gated entrypoints. For every retained feature, implement its contexts and handlers, declare its Cargo feature, and verify each supported build configuration.

Fix 1.1

The unused core modules, entrypoints, and feature gates were removed from the program.

Go back to Findings Summary

I7: Deny-all policy still permits transaction proposals

Impact:	Info	**Likelihood:**	N/A
Type:	Code quality	**Status:**	Fixed
Target:	programs/policy_deny_all/src/lib.rs		

Description

`policy_deny_all` accepts `Creation` while rejecting `Execution`, so installing it does not stop new transaction proposals:

Listing 92. Excerpt from Creation remains permitted while Execution is denied

```
13 ValidateContext::Creation(creation_ctx) => {
14     creation_ctx.validate_accounts(ctx.remaining_accounts)?;
15     Ok(())
16 }
17 ValidateContext::Execution(execution_ctx) => {
18     execution_ctx.validate_accounts(ctx.remaining_accounts)?;
19     Err(ErrorCode::Denied.into())
20 }
```

For example, if an operator installs the policy while vault transactions remain enabled, an ordinary signer can still create a proposal and pay its allocation rent, although the policy prevents its execution and other policy branches prevent fund movement.

Recommendation

Reject `Creation` if the policy must deny every operation; otherwise, rename and document the creation exception.

Disable vault transactions when incident response must block both proposal creation and execution.

Fix 1.1

The fixture source now documents that `Creation` remains allowed so tests can exercise denied execution. The out-of-scope website documentation still states that the policy blocks all transactions.

Go back to Findings Summary

I8: Duplicated core IDs can desynchronize authorization checks

Impact:	Info	**Likelihood:**	N/A
Type:	Logic error	**Status:**	Acknowledged
Target:	lib/fee_sdk/src/lib.rs, programs/policy_multisig/src/lib.rs		

Description

The fee SDK and multisig policy separately hard-code the core program ID:

Listing 93. Excerpt from Fee SDK core identity

```
18 pub const HYRO_PROTOCOL_PROGRAM_ID: Pubkey =
19
   anchor_lang::solana_program::pubkey!("HyroPk4VamtJfPfsxf77zRkMSx9FTV7FZqvH4
   JPup6fW");
```

Listing 94. Excerpt from Multisig core identity

```
25 pub const HYRO_PROTOCOL_PROGRAM_ID: Pubkey =
26
   anchor_lang::solana_program::pubkey!("HyroPk4VamtJfPfsxf77zRkMSx9FTV7FZqvH4
   JPup6fW");
```

These values currently match the core program ID, but a core-only identity change leaves fee-authority PDA derivation and multisig signer and owner checks bound to the previous ID. For example, after deploying the core under a new ID without updating the fee SDK, a legitimate fee call supplies a PDA derived from the new ID while the dispatcher expects one derived from the old ID and rejects the operation.

Recommendation

Re-export the core program ID through `hyro_sdk` for `policy_multisig`. Provide

`fee_sdk` with a cycle-free shared identity dependency or enforce equality with the core program ID at build time. Synchronize deployment configuration with all authorization-critical constants.

Acknowledgment 1.1

> `policy_multisig` is demo policy for documentation, out of audition scope
>
> — HyroTrader Team

Go back to Findings Summary

I9: Escrow bump comments contradict account declarations

Impact:	Info	**Likelihood:**	N/A
Type:	Code quality	**Status:**	Fixed
Target:	programs/hyro_protocol/src/funds.rs		

Description

Comments in both settlement handlers incorrectly state that `owner` is declared after the escrow authority, although it is declared before the authority. They therefore misattribute an extra escrow-bump calculation to an account-ordering problem and can cause developers to preserve an unnecessary workaround.

Listing 95. Excerpt from Escrow bump workaround explanation

```
970 // NOTE: derive the escrow-authority bump here instead of using
971 // `ctx.bumps.deposit_escrow_authority`. The escrow authority's seeds
972 // reference `owner`, which is declared after the authority in the
973 // accounts struct, so Anchor populates an incorrect bump in `ctx.bumps`
974 // even though seed validation passes.
```

Recommendation

Correct the comments in both settlement handlers. Reuse Anchor's `ctx.bumps` values when signing with the same seeds unless a confirmed framework bug requires recalculation.

Fix 1.1

Settlement now uses Anchor's canonical escrow-authority bumps, and the source comments describe the account ordering correctly. The out-of-scope protocol reference retains the obsolete workaround.

Go back to Findings Summary

I10: Escrow rent and surplus balances cannot be reclaimed

Impact:	Info	**Likelihood:**	N/A
Type:	Logic error	**Status:**	Acknowledged
Target:	programs/hyro_protocol/src/funds.rs		

Description

Deposit and withdrawal escrows use requester-funded `init_if_needed` token accounts, but settlement and cancellation close only the request data accounts. These flows move or burn only the recorded request amount, so any tokens transferred directly to an escrow remain there, and no core handler recovers surplus tokens or closes an empty token escrow.

Listing 96. Excerpt from Recorded cancellation amount

```
1055 let assets = ctx.accounts.deposit_request.assets;
1056 require!(
1057     ctx.accounts.escrow_token_account.amount >= assets,
1058     crate::vault::ErrorCode::InvalidDepositEscrow
1059 );
```

The inaccessible balance can include surplus deposit tokens, unburned surplus shares that remain in the mint supply, and the escrow account's rent. Reusing an escrow adds and consumes only each new request's recorded amount, so it does not recover an existing surplus.

Recommendation

Add owner-authorized recovery that transfers surplus tokens to an owner-controlled account of the correct mint while preserving amounts reserved for active requests.

Close empty token escrows through an escrow-authority-PDA-signed SPL

Token `CloseAccount` CPI and return their rent to the authorized owner.

Go back to Findings Summary

I11: Time and performance fees have no core accrual path

Impact:	Info	**Likelihood:**	N/A
Type:	Logic error	**Status:**	Reported
Target:	programs/hyro_protocol/src/funds.rs		

Description

The core's five fee calls emit only `Deposit`, `Withdraw`, `UseFunds`, or `ReturnFunds`, and never emit `TimeBased` or `Performance`. The time-only calculator therefore returns zero fees even after a full billing period because it accrues fees only for `TimeBased` operations:

Listing 97. Excerpt from time-only calculator operation check

```
324 let mut output = FeeCalculationOutput::default();
325
326 // Only calculate time-based fees for TimeBased operation
327 if input.operation == FeeOperation::TimeBased {
```

The fractional and all-in-one performance branches and their dispatcher caps are also unreachable. The unused `build_fee_input` helper provides no accrual entry point, and external callers cannot record the missing operations in the canonical ledger because the dispatcher requires the core's vault-share-signer PDA signature.

No supplied calculator reads the forwarded `last_fee_timestamp`. Stateful calculators use their own clocks, so resetting the core's `last_fee_collection` does not reset accrual. As a result, configured time and performance fees are not collected through the core, while the core timestamp does not control calculator accrual state.

Recommendation

Add authenticated core paths for `TimeBased` and `Performance` accrual, or disallow calculators that require unsupported operations.

Before enabling accrual, reconcile calculator clocks and performance high-water marks, calculate time fees from the intended assets-under-management basis and performance fees from net appreciation, and keep time-fee caps independent of performance profit.

Either enforce `last_fee_timestamp` as part of the calculator accrual contract or remove the unused timestamp plumbing.

Update 1.1

The new accrual instruction reaches time and performance fees, but unresolved period, high-water-mark, and cap behavior remains, and the supplied migration misses existing version-two fraction configurations.

Go back to Findings Summary

I12: Fee totals use inconsistent boundary arithmetic

Impact:	Info	**Likelihood:**	N/A
Type:	Code quality	**Status:**	Fixed
Target:	lib/fee_sdk/src/lib.rs, programs/fee_collection/src/lib.rs, programs/hyro_protocol/src/funds.rs		

Description

`FeeAmounts.total` uses saturating addition, while the ledger and core use checked arithmetic for fee liabilities:

Listing 98. Excerpt from Saturating fee aggregation

```
88 pub fn total(&self) -> u64 {
89     self.lp
90         .saturating_add(self.manager)
91         .saturating_add(self.protocol)
92         .saturating_add(self.performance)
93 }
```

For an LP fee of `u64::MAX`, a manager fee of 1, and zero other components, `total` returns `u64::MAX` instead of reporting that the aggregate is unrepresentable. Although no reachable overflow or operation-cap bypass is established, this behavior can conceal invalid hypothetical fee results from SDK integrators.

Claims also disregard the ledger's returned post-claim total. After the ledger call and token transfer at `programs/hyro_protocol/src/funds.rs:1816-1827`, the core recomputes its mirrored liability by subtracting the claim from its previous value:

Listing 99. Excerpt from Checked post-claim fee mirror

```
1831 let new_obtained = ctx
```

```
1832     .accounts
1833     .vault
1834     .obtained_fees
1835     .checked_sub(amount)
1836     .ok_or(crate::vault::ErrorCode::BalanceUnderflow)?;
1837 ctx.accounts.vault.update_obtained_fees(new_obtained)?;
```

The mirror remains correct while both records agree, but the independent calculation can hide disagreement at the arithmetic boundary.

Recommendation

Use checked addition for aggregate fee calculations. Authenticate and store the ledger's returned post-claim total, or verify that it equals the core's checked subtraction before updating the mirror. Keep the ledger debit and token payment atomic.

Fix 1.1

Fee aggregation now uses checked arithmetic, and claims verify both the returned ledger value and the core liability update before committing state.

Go back to Findings Summary

I13: Accrual fee cap permits the entire vault NAV

Impact:	Info	**Likelihood:**	N/A
Type:	Logic error	**Status:**	Reported
Target:	programs/fee_collection/src/lib.rs		

Description

The dispatcher caps `Deposit`, `Withdraw`, `UseFunds`, and `ReturnFunds` fees at `input.amount`, but caps accrual fees at the positive NAV, or zero when NAV is nonpositive:

Listing 100. Excerpt from Accrual cap selection

```
417 FeeOperation::TimeBased | FeeOperation::Performance => {
418     if input.total_balance > 0 {
419         input.total_balance as u64
420     } else {
421         0
422     }
423 }
```

The subsequent total-fee check therefore permits an accrual fee equal to the vault's entire positive NAV. The standalone time calculator also multiplies fees by elapsed periods without imposing a NAV-based or per-interval limit, so this whole-NAV bound does not enforce the documented operation-amount cap or a periodic fee budget.

The core currently emits only the four amount-based operation tags, leaving both accrual operations dormant. External callers cannot reach them because dispatch requires the core share-signer PDA, while proposed CPIs sign with the distinct vault-authority PDA. The defect is therefore a latent, overly broad bound that becomes effective if the core enables accrual.

Recommendation

Document the current whole-NAV bound and the unused accrual tags.

Before enabling accrual, enforce a dispatcher-side per-interval fee budget, such as a maximum NAV fraction, while preserving the existing operation-amount caps and producer-side checks on calculator returns.

Update 1.1

The fee cap applies independently to each operation and call, and an over-cap accrued backlog is rejected as a whole instead of being collected incrementally.

Go back to Findings Summary

I14: Manager registry fields do not track ongoing activity

Impact:	Info	**Likelihood:**	N/A
Type:	Code quality	**Status:**	Reported
Target:	programs/hyro_protocol/src/manager_registry.rs, programs/hyro_protocol/src/vault.rs		

Description

Manager registry statistics record issuance events but do not track subsequent lifecycle changes. Issuance increments the active-vault count and adds funded allocations to profile and registry AUM, while later balance changes and retirement do not adjust these values.

Listing 101. Excerpt from Issuance-only AUM update

```
1115 manager_profile.total_aum = manager_profile
1116     .total_aum
1117     .checked_add(allocation)
1118     .ok_or(ErrorCode::MathOverflow)?;
```

Claims update `Vault.total_fees_paid` without updating `ManagerProfile.total_fees_earned`. Indexer consumers therefore cannot interpret these fields as current AUM, active-vault totals, or realized manager revenue. For example, an allocation funded with 10,000 units still reports 10,000 units after trading losses reduce it to 6,000.

Suspension and blacklisting restrict future issuance but do not revoke existing manager assignments, allocations, or fees. The registry administrator controls both verification and issuance authorization, with no independent mechanism for ongoing revocation.

Recommendation

Define and name each statistic as either cumulative issuance or a live metric. Update live AUM, active-vault, and manager-fee counters at every applicable lifecycle transition.

Document that suspension and blacklisting affect only issuance, or enforce these states on each existing manager operation they must revoke.

Update 1.1

Issuance volume was renamed as cumulative, but `active_vaults` remains a cumulative count under a live-state name, realized-fee totals remain unwritten, and indexer schemas retain old fields.

Go back to Findings Summary

I15: Manager verification returns an inconsistent authorization error

Impact:	Info	**Likelihood:**	N/A
Type:	Code quality	**Status:**	Fixed
Target:	programs/hyro_protocol/src/manager_registry.rs		

Description

`VerifyManager` rejects unauthorized administrators through its account constraint before the handler runs, but the constraint omits `@ ErrorCode::UnauthorizedAdmin`:

Listing 102. Excerpt from VerifyManager authorization constraint

```
90 #[account(mut)]
91 pub admin: Signer<'info>,
92 #[account(mut)]
93 pub manager_profile: Box<Account<'info, ManagerProfile>>,
94 #[account(constraint = registry.admin == admin.key())]
95 pub registry: Box<Account<'info, ManagerRegistry>>,
```

Unauthorized callers therefore receive Anchor's generic raw-constraint error instead of the registry's "Unauthorized admin access" error. The handler repeats the check with `ErrorCode::UnauthorizedAdmin`, but failed constraints prevent unauthorized callers from reaching it. Verification remains rejected, but operators receive an inconsistent error that does not identify the wrong administrator wallet.

Recommendation

Attach `@ ErrorCode::UnauthorizedAdmin` to the `VerifyManager` registry constraint. Keep registry authorization errors consistent across registration and verification.

Fix 1.1

Manager verification now returns `UnauthorizedAdmin` when the registry administrator does not match. The out-of-scope IDL and generated clients retain obsolete error mappings.

Go back to Findings Summary

I16: Validation errors misidentify the failed constraint

Impact:	Info	**Likelihood:**	N/A
Type:	Code quality	**Status:**	Fixed
Target:	programs/hyro_protocol/src/funds.rs, programs/fee_collection_fractions/src/lib.rs, programs/fee_collection_time_based/src/lib.rs, programs/fee_collection_all_in_one/src/lib.rs		

Description

`RequestDeposit`, `ProcessDeposits`, `RequestWithdrawal`, and `ProcessWithdrawals` return `InvalidFeeCollectionProgram` when the policy program is incorrect, although other policy constraints use `InvalidPolicyProgram`.

The three calculator migrations return an authority error when the stored vault does not match its PDA, while the dispatcher reports the equivalent inconsistency as `VaultMismatch`:

Listing 103. Excerpt from migration vault consistency check

```
197 let mut cfg = hyro_sdk::load_account::<VaultFractionConfig>(&config)?;
198 require_keys_eq!(cfg.vault, ctx.accounts.vault.key(),
    ErrorCode::Unauthorized);
199 cfg.authority = vault.admin;
```

Changing the arbitrary migration payer cannot repair this preexisting inconsistent program-owned state. These errors misdirect diagnosis even though the checks fail closed and do not affect authorization or asset safety.

Recommendation

Use `InvalidPolicyProgram` for the four policy-program constraints.

Append `VaultMismatch` to each calculator's error enum without changing existing discriminants, and use it for migration vault consistency checks.

Fix 1.1

Policy and calculator constraints now return errors specific to the failed validation. The out-of-scope core IDL retains obsolete pre-offset error numbers.

Go back to Findings Summary

I17: Overlapping error codes misidentify rejected operations

Impact:	Info	**Likelihood:**	N/A
Type:	Code quality	**Status:**	Fixed
Target:	programs/hyro_protocol/src/lib.rs, programs/hyro_protocol/src/vault.rs, lib/hyro_sdk/src/validate.rs		

Description

The root, vault, manager-registry, and SDK `#[error_code]` enums use Anchor's default 6000 base, creating overlapping numeric error codes. Root `InvalidShareMint` and vault `InvalidAllocation` both encode as 6004, while the stored IDL maps 6004 only to `InvalidAllocation`:

Listing 104. Excerpt from IDL mapping for error 6004

```
3594 {
3595   "code": 6004,
3596   "name": "InvalidAllocation",
3597   "msg": "Invalid allocation amount."
3598 },
```

The SDK's 13 errors similarly occupy 6000 to 6012 and overlap core-local policy errors during the same policy execution. For example, SDK `InvalidPolicyAccount` and core `DepositsDisabled` both encode as 6010. Numeric-only decoding can therefore misidentify rejected operations, including propagated CPI errors, although the checks still reject them and Anchor logs retain their distinct names and messages.

Recommendation

Assign nonoverlapping ranges to the core enums, or consolidate them into one namespace. Reserve and publish a separate SDK range while preserving

public mappings where possible. Decode propagated CPI errors using the originating program and Anchor logs in addition to the numeric code.

Fix 1.1

Core and shared-library errors now use nonoverlapping numeric ranges. The out-of-scope IDL retains obsolete mappings, and downstream clients still need origin-aware decoding for propagated program errors.

Go back to Findings Summary

I18: Amount policies must decode core instructions

Impact:	Info	**Likelihood:**	N/A
Type:	Data validation	**Status:**	Reported
Target:	lib/hyro_sdk/src/validate.rs		

Description

The policy ABI serializes a fieldless operation enum without amounts or report values:

Listing 105. Excerpt from fieldless policy operation serialization

```
753 let mut instruction_data = Vec::with_capacity(8 + 32);
754 instruction_data.extend_from_slice(&VALIDATE_DISCRIMINATOR);
755 instruction_data.extend(operation.try_to_vec()?);
```

`UseFunds`, `ReturnFunds`, and `Report` provide the vault or state, signer, and Instructions sysvar. A policy that enforces amount or report-value constraints must therefore authenticate and decode the top-level core instruction from the sysvar, coupling the policy to core discriminators and Borsh argument layouts. Nested calls expose the outer instruction instead, so policies must reject unsupported instruction shapes. A core argument-layout change also requires coordinated policy decoder changes or causes policies to reject the new layout.

For example, requests to use 500 and 5,000 token base units both reach a policy as `UseFunds`; the operation arguments alone cannot distinguish whether either request exceeds a 1,000-unit limit.

Recommendation

Pass amounts, report updates, and operation identity through a versioned policy ABI, and coordinate the wire-format migration. Until migration,

document how policies must authenticate and decode the Instructions sysvar, and require them to reject unknown layouts and uninterpretable nested calls.

Update 1.1

Policy validation now receives an amount, but report validation receives only an unsigned absolute change and cannot distinguish direction, new value, or conditional report form.

Go back to Findings Summary

I19: Policy validation does not authenticate core provenance

Impact:	Info	**Likelihood:**	N/A
Type:	Code quality	**Status:**	Reported
Target:	programs/hyro_protocol/src/vault.rs, lib/hyro_sdk/src/validate.rs		

Description

Core policy validation uses plain `invoke` and forwards existing signer privileges without supplying a core-derived signature:

Listing 106. Excerpt from Policy validation invocation

```
755 let mut all_infos = Vec::with_capacity(2 + accounts.len());
756 all_infos.push(vault.clone());
757 all_infos.extend_from_slice(accounts);
758 all_infos.push(policy_program.clone());
759
760 solana_program::program::invoke(&instruction, &all_infos)?;
```

The vault-authority signature is supplied only to the subsequent approved-target CPI. Genuine vault accounts, correctly derived authority addresses, SDK account-presence checks, and actor signatures therefore do not authenticate the invoking program. Any caller can invoke public policy validation directly, so a successful validation proves policy approval but not core proposal creation or execution.

For example, a caller can invoke the stateless `policy_allow_any` Creation validation with the required account slots and receive success without creating a core proposal. An integrator that treats this result as proof of core proposal creation can accept an operation that the core never processed. Direct validation does not execute a core transaction or bypass its checks.

Recommendation

Document validation as public. Require policies to check account ownership, address relationships, actor signatures, and state-transition prerequisites. For core-only validation, define and require a dedicated core-derived validation signer.

Update 1.1

Documentation now calls policy validation public, but it incorrectly describes the vault-signer PDA as proof of a core-signed call even though the account is not required to sign.

Go back to Findings Summary

I20: Delay changes extend existing withdrawal locks

Impact:	Info	**Likelihood:**	N/A
Type:	Logic error	**Status:**	Reported
Target:	programs/hyro_protocol/src/vault.rs		

Description

Withdrawal requests store their creation time but not the applicable delay or maturity timestamp. Settlement applies the current administrator-configured delay to the original timestamp, so changing the delay retroactively changes when every pending request becomes eligible:

Listing 107. Excerpt from Settlement applies the current delay to pending requests

```
1167 let delay_i64 = i64::try_from(delay)
1168     .map_err(|_| error!(crate::vault::ErrorCode::WithdrawalNotReady))?;
1169 let ready_at = ctx
1170     .accounts
1171     .withdrawal_request
1172     .requested_at
1173     .checked_add(delay_i64)
1174     .ok_or_else(|| error!(crate::vault::ErrorCode::WithdrawalNotReady))?;
1175 require!(now >= ready_at, crate::vault::ErrorCode::WithdrawalNotReady);
```

For example, Alice requests a withdrawal when the delay is one day. If the administrator raises it to seven days before settlement, her request becomes eligible seven days after creation instead of after one day.

The setter accepts any `u64`, but values of at least `2^63` fail conversion and block all settlements while configured. Owners can still cancel requests and recover their recorded shares without waiting for the delay, and the administrator can separately pause redemptions.

Recommendation

Snapshot a checked maturity timestamp when each withdrawal is requested so pending requests retain their original terms. Reject delay values that settlement cannot convert. Use the explicit redemption pause for emergency suspension and document its administrator trust assumption separately.

Update 1.1

New requests store a fixed maturity, but legacy request migration applies the current delay and vault initialization can still install a delay above the setter limit.

Go back to Findings Summary

I21: Build caching omits shared Rust crates

Impact:	Info	**Likelihood:**	N/A
Type:	Data validation	**Status:**	Fixed
Target:	apps/protocol/project.json		

Description

The protocol build targets configure Nx cache inputs that omit shared Rust dependencies under `apps/protocol/lib`:

Listing 108. Excerpt from Protocol build cache inputs

```
"anchor": [
  "{projectRoot}/programs/**/*.rs",
  "{projectRoot}/programs/**/Cargo.toml",
  "{projectRoot}/Cargo.toml",
  "{projectRoot}/Cargo.lock",
  "{projectRoot}/Anchor.toml"
],
```

After a successful build, a change confined to a shared crate such as `hyro_sdk` does not invalidate an applicable cache entry. Nx can therefore restore stale compiled programs, IDLs, and type artifacts, causing subsequent testing or deployment to use code that does not include the updated source. The separate `default` inputs do not cover these explicitly configured targets.

Recommendation

Include shared Rust sources and manifests in both targets' `anchor` inputs through dependency-derived inputs or a source-only workspace pattern that covers future local crates. Invalidate affected cache entries and rebuild release artifacts from the complete source set.

Fix 1.1

The Nx Anchor input set now includes Rust library sources and manifests, including changes under `lib/` in cached protocol builds.

Go back to Findings Summary

I22: Signature lookup rejects valid transaction layouts

Impact:	Info	**Likelihood:**	N/A
Type:	Logic error	**Status:**	Fixed
Target:	lib/hyro_sdk/src/sigverify.rs		

Description

`find_signed_message` returns an error when it encounters an unsupported Ed25519 signature format instead of continuing to search later instructions:

Listing 109. Excerpt from Hard failure during signature lookup

```
87 // A malformed Ed25519 layout is a hard error: the bytes we would read
88 // might not be the bytes the precompile verified.
89 let (signer, message) = parse_ed25519_instruction(&instruction.data, index)
90     .ok_or(SigVerifyError::MalformedEd25519Instruction)?;
91 if &signer == expected_signer {
92     return Ok(message);
93 }
```

The helper supports only a single signature whose data resides in the verifier instruction, while Solana's native verifier also supports batched signatures and cross-instruction offsets. A transaction containing an unrelated unsupported Ed25519 instruction before a valid administrator voucher therefore fails despite carrying the required approval. Unsupported formats fail closed, and a third party cannot add an instruction to an already-signed transaction.

Recommendation

Continue searching after unsupported formats or parse all native verifier formats. Require a matching verified administrator signature, document the supported formats, and distinguish unsupported encodings from invalid signatures.

Fix 1.1

The Ed25519 instruction search now skips unsupported layouts and continues scanning for a supported instruction from the expected signer.

Go back to Findings Summary

I23: SDK always enables IDL-generation features

Impact:	Info	**Likelihood:**	N/A
Type:	Logic error	**Status:**	Fixed
Target:	lib/hyro_sdk/Cargo.toml		

Description

`hyro_sdk` declares `idl-build` as optional but unconditionally enables it on the `hyro_protocol` dependency together with `cpi`:

Listing 110. Excerpt from unconditional hyro_protocol features

```
28 [dependencies]
29 hyro_protocol = { path = "../../programs/hyro_protocol", features = ["idl-
   build", "cpi"] }
```

Because `hyro_protocol` forwards `idl-build` to Anchor language and SPL IDL generation, a policy developer enables those features even when depending on `hyro_sdk` with default features disabled and without selecting `idl-build`. The SDK's optional gate therefore cannot exclude the IDL-generation features from an ordinary build.

Recommendation

Remove `idl-build` from the unconditional `hyro_protocol` dependency features while retaining `cpi`. Add `hyro_protocol/idl-build` to the SDK's optional `idl-build` feature alongside `anchor-lang/idl-build`.

Fix 1.1

Ordinary `hyro_sdk` use now enables only the core CPI feature, while IDL generation is exposed through an explicit optional feature.

Go back to Findings Summary

I24: Unpinned token accounts produce misleading balance errors

Impact:	Info	**Likelihood:**	N/A
Type:	Data validation	**Status:**	Reported
Target:	programs/hyro_protocol/src/funds.rs		

Description

Deposit and withdrawal requests check an account's balance before validating that its mint matches the expected token:

Listing 111. Excerpt from balance check before mint validation

```
786 require!(
787     ctx.accounts.sender_token_account.amount >= assets,
788     crate::vault::ErrorCode::InsufficientFunds
789 );
```

A request that supplies a wrong-mint account with an insufficient balance therefore returns `InsufficientFunds`, which misidentifies the invalid token account. If the balance is sufficient, the later SPL Token transfer rejects the mint mismatch. Withdrawal and fee-payment recipients and deposit-cancellation refund accounts also lack early mint validation. Mint-bound vault and escrow accounts prevent wrong-token transfers, and refunds remain bound to the request owner.

Recommendation

Validate that every input, recipient, and refund account holds the expected underlying or share token before checking its balance or transferring funds. Preserve request-owner authorization for refunds.

Update 1.1

Deposit paths now check token mints early, but withdrawal requests, payout recipients, cancellation accounts, and fee recipients still lack equivalent early mint validation.

Go back to Findings Summary

I25: Child policy validation receives unwritten vault state

Impact:	Info	**Likelihood:**	N/A
Type:	Code quality	**Status:**	Fixed
Target:	lib/hyro_sdk/src/validate.rs, programs/hyro_protocol/src/vault.rs		

Description

Child vault creation calls the parent policy before writing the child's manager, parent, allocation, and other settings. The policy receives the unwritten child account and the manager separately, although the SDK states that the policy can deserialize the child fields during validation:

Listing 112. Excerpt from SDK child-account contract

```
822 /// The child `Vault` account being created. It is `init`ed by the core
    before
823 /// the CPI, so a policy can deserialize it to read
    `manager`/`parent_vault`.
```

A policy therefore cannot validate proposed terms that exist only in the child state, such as enforcing a 10,000 allocation limit, before approving creation.

Recommendation

Correct the SDK comment. Pass the proposed creation terms explicitly to the policy, or populate and serialize the relevant child fields before validation, as `create_tx` does for proposals. Rely on transaction rollback when the policy rejects the populated state. Retain the separate manager account and verify that it matches the child state.

Fix 1.1

Child vault state is now fully initialized and serialized before the parent policy validates the proposed issuance.

Go back to Findings Summary

I26: Vault lifecycle documentation and state are inconsistent

Impact:	Info	**Likelihood:**	N/A
Type:	Code quality	**Status:**	Fixed
Target:	programs/hyro_protocol/src/vault.rs		

Description

The `Vault` configuration comment describes deposit settings as immutable even though administrators can change them:

Listing 113. Excerpt from Configuration comment

```
110 // Vault configuration (set at init, immutable after)
111 pub config: Option<VaultConfig>,
```

An integration that relies on this comment may cache the settings and continue offering deposits after an administrator disables them, causing the protocol to reject those operations.

The proposal-size comment states that the checks prevent excessive allocation, but Anchor allocates the account before the handler runs:

Listing 114. Excerpt from Allocation-limit comment and checks

```
775 // AUDIT M-10: bound the payload so `transaction_space` cannot be used to
776 // request an arbitrarily large account at the payer's expense.
777 require!(data.len() <= MAX_TX_DATA_LEN, ErrorCode::TransactionTooLarge);
778 require!(accs.len() <= MAX_TX_ACCOUNTS, ErrorCode::TransactionTooLarge);
```

An oversized proposal reverts the allocation and rent but still attempts the allocation.

Execution also sets `did_execute` before closing the transaction account.

Closure prevents replay, while the flag disappears with the account and therefore does not retain an execution record.

Recommendation

Correct the comments to describe mutable settings, their defaults, and the timing and effect of the proposal-size checks.

Document account closure as the replay protection mechanism.

Remove `did_execute` through an account-layout migration, or document that it remains for layout compatibility.

Fix 1.1

The in-scope vault and transaction comments now describe mutable configuration, payload allocation, and closure-based replay protection. Out-of-scope references still describe the removed `did_execute` flag.

Go back to Findings Summary

I27: Vault state omits its virtual balance mode

Impact:	Info	**Likelihood:**	N/A
Type:	Data validation	**Status:**	Fixed
Target:	programs/hyro_protocol/src/vault.rs		

Description

Vault creation uses `virtual_balance` to select whether a child vault receives tokens from its parent or records demo equity without receiving tokens, but the vault record does not store this mode:

Listing 115. Excerpt from Virtual balance controls creation without persisting the mode

```
1027 if virtual_balance {
1028     // Demo trading: the vault anchors off-chain equity and holds no
     tokens.
1029     child_vault.update_onchain_balance(BalanceUpdate::NewValue(0))?;
1030
     child_vault.update_offchain_balance(BalanceUpdate::NewValue(allocation))?
     ;
1031 } else {
1032
     child_vault.update_onchain_balance(BalanceUpdate::NewValue(allocation))?;
1033     child_vault.update_offchain_balance(BalanceUpdate::NewValue(0))?;
1034 }
```

A virtual child with 10,000 units of demo equity and a funded child with 10,000 units held off-chain can each have a zero on-chain balance, a 10,000-unit off-chain balance, and no share mint at creation. Software reading the vault record therefore cannot distinguish demo equity from actual assets without verified creation history or another external source.

Recommendation

Store and expose an explicit `Virtual` or `Funded` mode when creating a vault.

Determine the mode of existing vaults from verified creation history. Keep virtual allocations excluded from manager and registry AUM.

Fix 1.1

Vault mode is now explicit on chain, migration derives it for existing vaults, and virtual allocations remain excluded from AUM. The out-of-scope indexer and website projections omit the mode.

Go back to Findings Summary

Report Revision 1.1

Revision Team

NAME	POSITION
Jan Kalivoda	Lead Auditor
Naoki Yoshida	Auditor
Michal Převrátil	Auditor
Josef Gattermayer, Ph.D.	Audit Supervisor

System Overview

Revision 1.1 made internal-movement and report fees optional and disabled them by default (H1), bound fee configuration to the live vault administrator (H2), and charged settlement fees to each transacting party (H3). Fee CPIs now use a custody-less authority (W7), and child vault reports now use their parent vault's oracle (L2). The revision also introduced versioned account layouts and migrations, but legacy serialization slack remained unresolved (I5).

Trust Model

Revision 1.1 removed fee-configuration control from former vault administrators (H2) and removed independent oracle appointment for child vaults (L2). Users must still trust the current vault administrator to select the policy, oracle, and fee configuration; the designated oracle to report off-chain equity faithfully (M8); and every manager authorized by the active policy to safeguard staged capital (W11). An unsafe policy can still expose funds to unauthorized callers (C1), and on-chain configuration cannot prove

that reported assets exist at the off-chain trading venue.

Appendix A: How to cite

Cite this document as:

> ack3, Security Review | HyroTrader: Hyro Protocol, October 8, 2026.

Appendix B: Review Scope

FILE	REVIEW TYPE
apps/protocol/programs/hyro_protocol/src/funds.rs	Manual review
apps/protocol/programs/hyro_protocol/src/vault.rs	Manual review
apps/protocol/programs/hyro_protocol/src/lib.rs	Manual review
apps/protocol/programs/hyro_protocol/src/helpers.rs	Manual review
apps/protocol/programs/hyro_protocol/src/constants.rs	Manual review
apps/protocol/lib/hyro_sdk/src/validate.rs	Manual review
apps/protocol/lib/hyro_sdk/src/sigverify.rs	Manual review
apps/protocol/lib/hyro_sdk/src/migrate.rs	Manual review
apps/protocol/lib/hyro_sdk/src/lib.rs	Manual review
apps/protocol/lib/hyro_sdk/src/states.rs	Manual review
apps/protocol/lib/fee_sdk/src/lib.rs	Manual review
apps/protocol/programs/fee_collection/src/lib.rs	Manual review
apps/protocol/programs/policy_checkout/src/lib.rs	Manual review
apps/protocol/programs/policy_checkout/src/message.rs	Manual review
apps/protocol/programs/hyro_protocol/src/manager_registry.rs	ack3 AI scan
apps/protocol/programs/hyro_protocol/src/states.rs	ack3 AI scan

FILE	REVIEW TYPE
apps/protocol/programs/fee_collection_fractions/src/lib.rs	ack3 AI scan
apps/protocol/programs/fee_collection_time_based/src/lib.rs	ack3 AI scan
apps/protocol/programs/fee_collection_all_in_one/src/lib.rs	ack3 AI scan
apps/protocol/programs/policy_challenges/src/lib.rs	ack3 AI scan
apps/protocol/programs/policy_challenges/src/policy.rs	ack3 AI scan
apps/protocol/programs/policy_challenges/src/state.rs	ack3 AI scan
apps/protocol/programs/policy_challenges/src/errors.rs	ack3 AI scan
apps/protocol/programs/policy_sig_gate/src/lib.rs	ack3 AI scan
apps/protocol/programs/policy_multisig/src/lib.rs	ack3 AI scan
apps/protocol/programs/policy_manager_access/src/lib.rs	ack3 AI scan
apps/protocol/programs/policy_owners/src/lib.rs	ack3 AI scan
apps/protocol/programs/policy_limit_transfer/src/lib.rs	ack3 AI scan
apps/protocol/programs/revenue_generator/src/lib.rs	ack3 AI scan
apps/protocol/programs/policy_withdrawal_window/src/lib.rs	ack3 AI scan

FILE	REVIEW TYPE
apps/protocol/programs/policy_deny_all/src/lib.rs	ack3 AI scan
apps/protocol/programs/policy_allow_any/src/lib.rs	ack3 AI scan
apps/protocol/programs/policy_admin_only/src/lib.rs	ack3 AI scan
apps/protocol/programs/dropper/src/lib.rs	ack3 AI scan

Thank you.

COMPANY	ACK3 LTD
WEB	ack3.ai
X	@ack3ai
MAIL	hello@ack3.ai