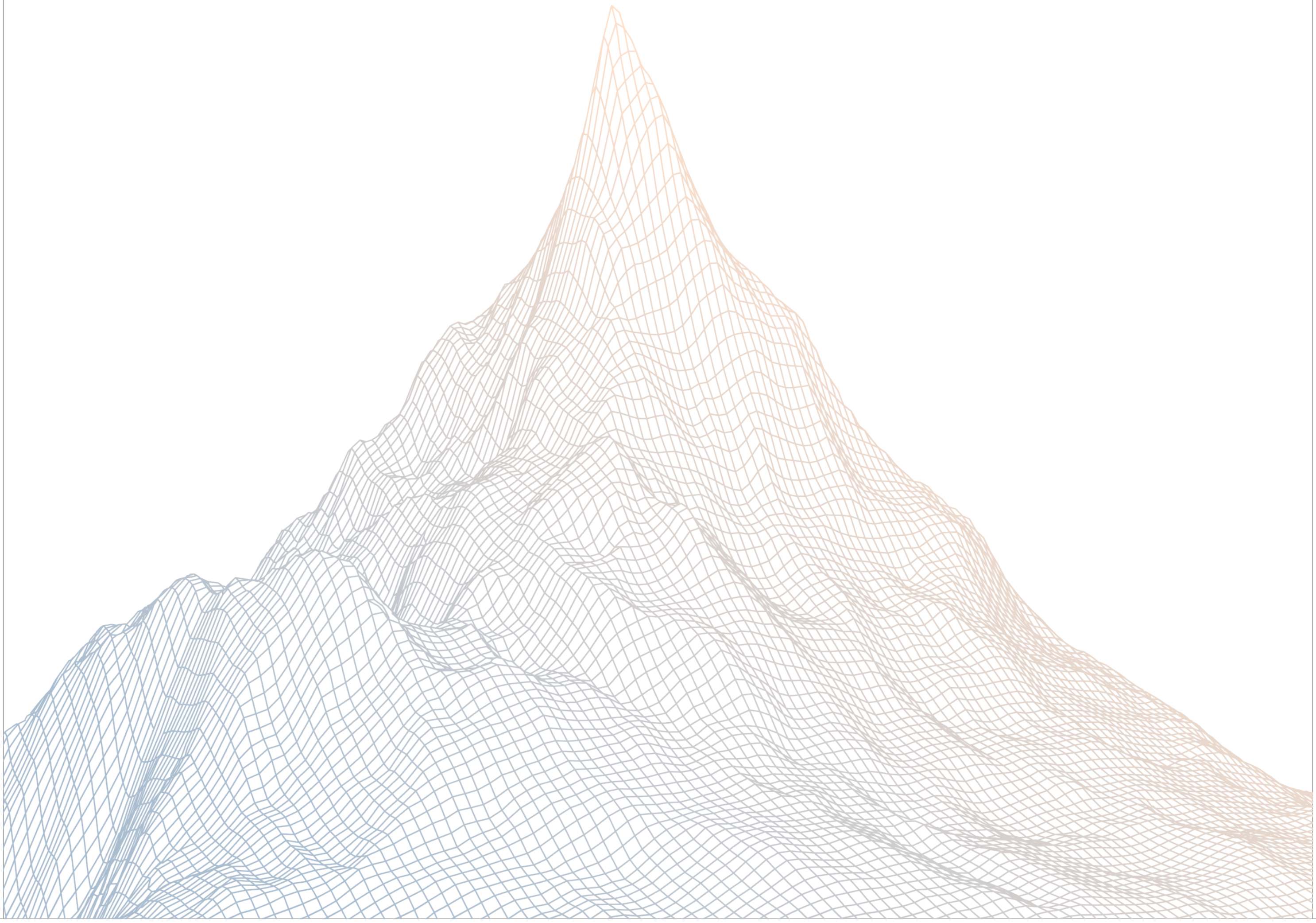
Zenith

Harmonix Finance

Smart Contract Security Assessment

VERSION 1.1



AUDIT DATES:	September 19th to September 24th, 2025
AUDITED BY:	Matte adriro

Contents

1 Introduction

1.1 About Zenith

Zenith assembles auditors with proven track records: finding critical vulnerabilities in public audit competitions.

Our audits are carried out by a curated team of the industry's top-performing security researchers, selected for your specific codebase, security needs, and budget.

Learn more about us at https://zenith.security.

1.2 Disclaimer

This report reflects an analysis conducted within a defined scope and time frame, based on provided materials and documentation. It does not encompass all possible vulnerabilities and should not be considered exhaustive.

The review and accompanying report are presented on an "as-is" and "as-available" basis, without any express or implied warranties.

Furthermore, this report neither endorses any specific project or team nor assures the complete security of the project.

1.3 Risk Classification

SEVERITY LEVEL	IMPACT: HIGH	IMPACT: MEDIUM	IMPACT: LOW
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

2 Executive Summary

2.1 About Harmonix

Harmonix Finance is a DeFi platform that combines advanced hedge fund-grade strategies with native blockchain technology to optimize yield generation and liquidity efficiency. Designed for both retail and institutional investors, Harmonix offers tools to generate sustainable returns on assets like ETH, BTC, stablecoins, and DeFi positions such as staked ETH, restaked ETH, PT Pendle, and Hyperliquid tokens.

2.2 Scope

The engagement involved a review of the following targets:

Target	core-contracts
Repository	https://github.com/harmonixfi/core-contracts
Commit Hash	`91d859ca28557123575debfed15840d6db1babca`

Files	`balanceContract.sol`
	`fundContract.sol`
	`fundStorage.sol`

2.3 Audit Timeline

September 19, 2025	Audit start
September 24, 2025	Audit end
October 3, 2025	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	0
High Risk	1
Medium Risk	3
Low Risk	6
Informational	0
Total Issues	**10**

3 Findings Summary

ID	Description	Status
M-1	NAV calculation incorrectly excludes network cost leading to understatement of vault assets	Resolved
M-2	The network cost parameter is never persisted to storage	Resolved
M-3	Inaccurate fee handling	Resolved
H-1	Pending withdrawals are not accounted for in the price per share calculation	Acknowledged
L-1	Account balance doesn't consider locked shares	Resolved
L-2	Network cost is paid for each partial withdrawal	Resolved
L-3	Incorrect withdrawPoolAmount check in _withdraw()	Resolved
L-4	Capacity should be denominated in assets	Resolved
L-5	Incorrect role in executeBatchActions()	Resolved
L-6	Incorrect return values for mint() and withdraw()	Resolved

4 Findings

4.1 Medium Risk

A total of 3 medium risk findings were identified.

[M-1] NAV calculation incorrectly excludes network cost leading to understatement of vault assets

`SEVERITY`: Medium	`IMPACT`: Medium
`STATUS`: Resolved	`LIKELIHOOD`: Medium

Target

- fundContract.sol

Description

The `_withdraw()` function contains a critical accounting error in how it handles the network cost during withdrawals. The function deducts the `networkCost` from the user's received assets and then mints shares representing both the management fee and network cost to the fee receiver:

```
sendAmount - networkCost  // User receives less assets
assetsToMint = managementFee + networkCost  // Shares minted for both fees
```

The core issue lies in the subsequent call to `_updateVaultState()`. While shares are minted for both the management fee and network cost (increasing the total supply), the NAV adjustment only accounts for `sharesWithdrawAmount - totalFees`, where `totalFees` includes only the management fee but excludes the network cost.

This creates an accounting discrepancy: the vault mints shares representing the network cost (diluting existing shareholders), but doesn't properly reflect this value in the NAV calculation. Since shares are minted for the network cost, those shares represent a claim on vault assets, and the corresponding value should remain part of the total assets. By not including the network cost in the NAV adjustment, the vault understates its total assets value.

The highest impact scenario occurs during periods of high withdrawal activity. Each withdrawal that incurs a network cost will progressively understate the NAV, leading to an accumulated discrepancy between the actual vault assets and the reported NAV. This affects all shareholders as it misrepresents the true value per share, potentially leading to incorrect pricing for subsequent deposits and withdrawals.

Recommendations

Include the network cost when calling `_updateVaultState()` to ensure the NAV properly reflects all minted shares:

```
updateVaultState(vaultState, sharesWithdrawAmount, totalFees);
updateVaultState
  (vaultState, sharesWithdrawAmount, totalFees + networkCost);
```

Harmonix: Since we don't charge management fees in the withdrawal flow, the NAV should be updated by subtracting sharesWithdrawAmount. Resolved with @8ca94abfbb...

Zenith: Verified. The implementation subtracts `sharesWithdrawAmount` from the NAV. Fees have been removed, and the network cost is transferred to the management fee receiver, instead of minting shares to represent this value.

[M-2] The network cost parameter is never persisted to storage

SEVERITY: Medium	IMPACT: Medium
STATUS: Resolved	LIKELIHOOD: Medium

Target

- fundContract.sol

Description

The `setVaultSetting()` function in the `VaultStore` library contains a storage bug where the `networkCost` parameter is not persisted to storage despite being part of the `VaultSetting` struct. This occurs specifically in the implementation of `setVaultSetting()`.

When examining the function, all vault setting parameters are properly stored to the `fundStorage` contract except for `networkCost`. The function stores `minimumSupply`, `capacity`, `performanceFeeRate`, `managementFeeRate`, `managementFeeReceiver`, and `performanceFeeReceiver` but omits the crucial storage operation for `vaultSetting.networkCost` using the `NETWORK_COST` key.

This is particularly problematic because the `getVaultSetting()` function at line 141 correctly retrieves the `networkCost` value from storage using `fundStorage.getUint256(keccak256(abi.encode(key, NETWORK_COST)))`, but since `setVaultSetting()` never stores this value, it will always return zero regardless of what value was passed during vault setup or updates.

The `networkCost` parameter is used throughout the vault contract for calculating withdrawal fees in the `_withdraw()` function.

When users withdraw funds, the contract attempts to deduct network costs from their withdrawal amount, but since `networkCost` is always zero, no network fees are collected. This results in the vault operator bearing all network costs for user withdrawals, leading to direct financial losses and improper fee collection that could accumulate significantly over time with high withdrawal volume.

Recommendations

Add the missing storage operation for `networkCost` in the `setVaultSetting()` function.

Harmonix: Resolved with @8ca94abfbb...

Zenith: Verified.

[M-3] Inaccurate fee handling

`SEVERITY:` Medium	`IMPACT:` Low
`STATUS:` Resolved	`LIKELIHOOD:` High

Target

- fundContract.sol

Description:

Management and performance fees are minted as vault shares and are calculated in a way that would yield less than the intended number of assets.

The implementation calculates the minted shares by converting the fee amount using `convertToShares()`. For example, let's assume we have 100 assets and 100 shares, and a fee rate of 10%:

1. `feeAmount = 100 * 10% = 10`
2. `sharesToMint = convertToShares(feeAmount) = 10 * 100 / 100 = 10`
3. 10 shares are minted to the fee receiver. Total supply is now 110.
4. Fee receiver now has `10 shares = convertToAssets(10) = 10 * 100 / 110 = 9 assets`, instead of the expected 10.

Recommendations:

The correct calculation should be:

```
sharesToMint = feeAmount * totalSupply / (PPS * totalSupply - feeAmount)
```

Harmonix: Resolved with @8ca94abfbb...

Zenith: Verified. Fee shares are now calculated as `fee * totalSupply() / (totalAssets() - fee)`.

[H-1] Pending withdrawals are not accounted for in the price per share calculation

`SEVERITY:` High	`IMPACT:` Medium
`STATUS:` Acknowledged	`LIKELIHOOD:` High

Target

- fundContract.sol

Description:

When a user initiates a withdrawal, the corresponding amount of assets for the requested shares is calculated based on the current price per share.

However, the amount of redeemed shares and assets is still counted as part of the vault's equity. Shares are locked in the `lockedShares` mapping but still counted as part of the total supply, and assets are still counted as part of the NAV.

Consequently, queued withdrawals will be considered in the implementation of `updateNav()`, which updates the price per share based on the NAV and total share supply. Any gain or loss in the vault would also be affected by these pending withdrawals, which have their price per share frozen.

Recommendations:

Remove locked shares and pending withdrawal amounts from the price per share calculation.

Harmonix: Acknowledged.

Zenith: Note that the impact depends on the proportion of assets pending withdrawal relative to the vault's equity. A large amount of pending withdrawals could significantly affect the share price.

4.2 Low Risk

A total of 6 low risk findings were identified.

[L-1] Account balance doesn't consider locked shares

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target

- fundContract.sol

Description:

As users initiate withdrawals, their shares are locked until they are redeemed. While users cannot transfer locked shares due to the override of `_update()`, their balances still account for the locked shares.

In particular, the implementation of `initiateWithdrawal()` checks the user's `balanceOf()` without taking into account their locked shares. This could lead to re-issuing withdrawals for the same shares if not for the pending withdrawal check.

```
458:     function initiateWithdrawal(
459:         uint256 _shares,
460:         uint256 _minAssetsOut
461:     ) external nonReentrant {
462:         require(!paused, "VAULT_PAUSED");
463:         require(balanceOf(msg.sender) >= _shares, "INV_SHARES");
```

Recommendations:

Change the requirement in `initiateWithdrawal()` to account for the locked shares.

```
require(balanceOf(msg.sender) - lockedShares[user] >= _shares,
    "INV_SHARES");
```

Harmonix: Resolved with @8ca94abfbb...

Zenith: Verified.

[L-2] Network cost is paid for each partial withdrawal

`SEVERITY`: Low	`IMPACT`: Low
`STATUS`: Resolved	`LIKELIHOOD`: Medium

Target

- fundContract.sol

Description:

In the `_withdraw()` function, the `networkCost` is subtracted from the `sendAmount` to the user.

Since users can execute partial withdrawals of their total queued shares, this means they will pay the fixed network cost each time, potentially paying it multiple times for a single withdrawal request.

Recommendations:

Consider adjusting the logic so that the cost is applied only once. The network cost can be subtracted at the moment the withdrawal request is initiated.

Harmonix: We introduced a new flag `isCollectNetworkCost`, which is used to check whether a withdrawal has already collected the network cost, ensuring that each withdrawal is charged only once. Resolved with @8ca94abfbb... and @a7fc401f45...

Zenith: Verified. Note that users can intentionally make a small withdrawal, lower than the network cost, to avoid paying it and have the `isCollectNetworkCost` flag toggled.

[L-3] Incorrect `withdrawPoolAmount` check in `_withdraw()`

`SEVERITY`: Low	`IMPACT`: Low
`STATUS`: Resolved	`LIKELIHOOD`: Low

Target

- fundContract.sol

Description:

During withdrawals, the implementation checks that the `withdrawPoolAmount` is enough to cover the transferred amount (line 764).

```
761:         uint256 sendAmount = sharesWithdrawAmount - totalFees;
762:
763:         require(
764:             vaultState.withdrawPoolAmount >= sendAmount,
765:             "EXCEED_WD_POOL_CAP"
766:         );
767:
768:         uint256 networkCost = sendAmount > vaultSetting.networkCost
769:             ? vaultSetting.networkCost
770:             : 0;
771:
772:         _updateUserWithdrawal(
773:             msg.sender,
774:             userWithdraw,
775:             _shares,
776:             sharesWithdrawAmount
777:         );
778:
779:         lockedShares[_owner] -= _shares;
780:         _burn(msg.sender, _shares);
781:
782:         TransferHelper.safeTransfer(
783:             asset(),
784:             _receiver,
785:             sendAmount - networkCost
786:         );
787:
```

```
788:          uint256 assetsToMint = managementFee + networkCost;
789:          uint256 sharesToMint = convertToShares(assetsToMint);
790:          _mint(vaultSetting.managementFeeReceiver, sharesToMint);
791:          TransferHelper.safeTransfer(
792:              asset(),
793:              balanceContract,
794:              totalFees + networkCost
795:          );
```

Note that `sendAmount` is calculated as the withdrawal amount (`sharesWithdrawAmount`) minus the fees (`totalFees`).

However, the vault ends up transferring `sendAmount - networkCost` (line 785) and `totalFees + networkCost` (line 794), which adds up to `sendAmount - networkCost + totalFees + networkCost = sendAmount + totalFees = sharesWithdrawAmount`.

Recommendations:

Change the check to `require(vaultState.withdrawPoolAmount >= sharesWithdrawAmount)`.

Harmonix: Resolved with 8ca94abfbb...

Zenith: Verified. Fixed the condition as recommended.

[L-4] Capacity should be denominated in assets

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target

- fundContract.sol

Description:

The implementation compares the deposit amount with the remaining capacity to validate the action.

```
407:         uint256 remainingCapacity = vaultSetting.capacity
    - totalSupply();
408:         require(_amount <= remainingCapacity, "INV_CAPACITY");
```

Note that `remainingCapacity` is calculated using shares (`totalSupply()`), but the comparison is done using `_amount`, which are assets.

Recommendations:

Calculate the remaining capacity using `totalAssets()`.

Harmonix: Resolved with 8ca94abfbb...

Zenith: Verified. Fixed as recommended.

[L-5] Incorrect role in `executeBatchActions()`

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target

- balanceContract.sol

Description:

The `executeBatchActions()` function is guarded by the `CONTROLLER` role, but it should be the `OPERATOR` role, the same as in `executeAction()`.

Recommendations:

Change `Role.CONTROLLER` to `ROLE.OPERATOR` in `executeBatchActions()`.

Harmonix: Resolved with 8ca94abfbb...

Zenith: Verified.

[L-6] Incorrect return values for `mint()` and `withdraw()`

`SEVERITY:` Low	`IMPACT:` Low
`STATUS:` Resolved	`LIKELIHOOD:` Low

Target

- fundContract.sol

Description:

The `mint()` function returns the value of `_deposit()`, which are issued shares, and should return assets.

Similarly, `withdraw()` returns the value of `_withdraw()`, which are the assets, and should return the redeemed shares.

Note also that implementation of `withdraw()` is not ERC4626 compliant, as fees and associated costs are taken out of the amount of assets, while the standard requires transferring exactly the requested amount.

Recommendations:

In `mint()` return `assets`, and in `withdraw()` return `shares`.

Harmonix: Resolved with 8ca94abfbb...

Zenith: Verified.