

September 10, 2025

SMART CONTRACT AUDIT REPORT

Ultra Yield
Vault Contracts



omniscia.io



info@omniscia.io



Online report: [ultra-yield-vault-contracts](#)

Omniscia.io is one of the fastest growing and most trusted blockchain security firms and has rapidly become a true market leader. To date, our team has collectively secured over 370+ clients, detecting 1,500+ high-severity issues in widely adopted smart contracts.

Founded in France at the start of 2020, and with a track record spanning back to 2017, our team has been at the forefront of auditing smart contracts, providing expert analysis and identifying potential vulnerabilities to ensure the highest level of security of popular smart contracts, as well as complex and sophisticated decentralized protocols.

Our clients, ecosystem partners, and backers include leading ecosystem players such as L'Oréal, Polygon, AvaLabs, Gnosis, Morpho, Vesta, Gravita, Olympus DAO, Fetch.ai, and LimitBreak, among others.

To keep up to date with all the latest news and announcements follow us on twitter @omniscia_sec.



omniscia.io



info@omniscia.io

Online report: [ultra-yield-vault-contracts](#)

Vault Contracts Security Audit

Audit Report Revisions

Commit Hash	Date	Audit Report Hash
b4cf59242d	August 24th 2025	5c55e478f9
28f2785396	September 3rd 2025	e02b460316
2d9651dbd7	September 8th 2025	37c6ba829e
2d9651dbd7	September 10th 2025	be347e896e

Audit Overview

We were tasked with performing an audit of the Ultra Yield codebase and in particular their vault contract implementations.

The system implements two types of **EIP-4626** vaults; one integrates as a wrapper contract on top of another vault whereas the other functions as the base vault implementation of the system.

Both implementations adhere to the **EIP-4626** standard as well as the **EIP-7540** and **EIP-7575** standards to facilitate a delayed withdrawal mechanism via redemption requests.

Over the course of the audit, we identified certain EIP compliancy issues, a cumulative truncation problem, as well as a significant issue in how deposits are handled in the `UltraVault` contract.

We advise the Ultra Yield team to closely evaluate all minor-and-above findings identified in the report and promptly remediate them as well as consider all optimizational exhibits identified in the report.

Post-Audit Conclusion

The Ultra Yield team iterated through all findings within the report and provided us with a revised commit hash to evaluate all exhibits on.

We evaluated all alleviations performed by Ultra Yield and have identified that a certain exhibit has not been adequately dealt with. We advise the Ultra Yield team to revisit the following exhibit: **BCA-01M**

Additionally, the following **informational** findings remain partially addressed and should be revisited: **UVR-04C**, **BCA-02C**

Post-Audit Conclusion (2d9651dbd7)

The Ultra Yield team proceeded with updating their codebase to rectify the three remaining issues outlined in the previous summary.

We evaluated the alleviations provided for those items and assessed them as fully addressed.

We consider all outputs of the audit report properly consumed by the Ultra Yield team with no outstanding remediative actions remaining.

Audit Synopsis

Severity	Identified	Alleviated	Partially Alleviated	Acknowledged
<div><div></div>Unknown</div>	0	0	0	0
<div><div></div>Informational</div>	25	24	0	1
<div><div></div>Minor</div>	4	3	0	1
<div><div></div>Medium</div>	3	2	0	1
<div><div></div>Major</div>	0	0	0	0

During the audit, we filtered and validated a total of **3 findings utilizing static analysis** tools as well as identified a total of **29 findings during the manual review** of the codebase. We strongly recommend that any minor severity or higher findings are dealt with promptly prior to the project's launch as they can introduce potential misbehaviours of the system as well as exploits.

- **Scope**
- **Compilation**
- **Static Analysis**
- **Manual Review**
- **Code Style**

Scope

The audit engagement encompassed a specific list of contracts that were present in the commit hash of the repository that was in scope. The tables below detail certain meta-data about the target of the security assessment and a navigation chart is present at the end that links to the relevant findings per file.

Target

- Repository: **<https://github.com/UltraYield/contracts>**
- Commit: **b4cf59242df2566e10e0670040658f8d851cba1e**
- Language: Solidity
- Network: Ethereum
- Revisions: **b4cf59242d, 28f2785396, 2d9651dbd7**

Contracts Assessed

File	Total Finding(s)
src/utils/AddressUpdates.sol (AUS)	1
src/vaults/BaseControlledAsyncRedeem.sol (BCA)	7
src/utils/FixedPointMathLib.sol (FPM)	0
src/oracles/OracleAdmin.sol (OAN)	1
src/vaults/accounting/RedeemQueue.sol (RQE)	1
src/vaults/UltraVault.sol (UVT)	4
src/vaults/UltraFeeder.sol (UFR)	2
src/oracles/UltraVaultOracle.sol (UVO)	6
src/oracles/UltraVaultRateProvider.sol (UVR)	5
src/oracles/VaultPriceManager.sol (VPM)	5

Compilation

The project utilizes `foundry` as its development pipeline tool, containing an array of tests and scripts coded in Solidity.

To compile the project, the `build` command needs to be issued via the `forge` CLI tool:

```
BASH
```

```
forge build
```

The `forge` tool automatically selects Solidity version `0.8.28` based on the version specified within the `foundry.toml` file.

The project contains discrepancies with regards to the Solidity version used as the `pragma` statements of the contracts are open-ended (`>=0.8.0`).

We advise them to be locked to `0.8.28` (`=0.8.28`), the same version utilized for our static analysis as well as optimizational review of the codebase.

During compilation with the `foundry` pipeline, no errors were identified that relate to the syntax or bytecode size of the contracts.

Static Analysis

The execution of our static analysis toolkit identified **41 potential issues** within the codebase of which **38 were ruled out to be false positives** or negligible findings.

The remaining **3 issues** were validated and grouped and formalized into the **3 exhibits** that follow:

ID	Severity	Addressed	Title
BCA-01S	<div><div></div>Informational</div>	<div><div>✓</div>Yes</div>	Inexistent Sanitization of Input Address
OAN-01S	<div><div></div>Informational</div>	<div><div>✓</div>Yes</div>	Inexistent Sanitization of Input Address
RQE-01S	<div><div></div>Informational</div>	<div><div>✓</div>Yes</div>	Inexistent Visibility Specifier

Manual Review

A **thorough line-by-line review** was conducted on the codebase to identify potential malfunctions and vulnerabilities in Ultra Yield's vault implementations.

As the project at hand implements an EIP-4626 vault system, intricate care was put into ensuring that the **flow of funds within the system conforms to the specifications and restrictions** laid forth within the protocol's specification.

We validated that **all state transitions of the system occur within sane criteria** and that all rudimentary formulas within the system execute as expected. We **pinpointed a significant dilution vulnerability** within the system which could have had **moderate ramifications** to all new depositors in the `UltraVault` system; for more information, kindly consult the relevant medium-severity exhibit of the contract within the report.

Additionally, the system was investigated for any other commonly present attack vectors such as re-entrancy attacks, mathematical truncations, logical flaws and **ERC / EIP** standard inconsistencies. The documentation of the project was satisfactory to the extent it need be.

A total of **29 findings** were identified over the course of the manual review of which **9 findings** concerned the behaviour and security of the system. The non-security related findings, such as optimizations, are included in the separate **Code Style** chapter.

The finding table below enumerates all these security / behavioural findings:

ID	Severity	Addressed	Title
BCA-01M	<div><div></div>Informational</div>	<div><div></div>Yes</div>	Incompatibility of System with EIP-777 Assets
BCA-02M	<div><div></div>Minor</div>	<div><div></div>Acknowledged</div>	Improper Accounting Error Drift
BCA-03M	<div><div></div>Minor</div>	<div><div></div>Yes</div>	Incorrect Event Emission
BCA-04M	<div><div></div>Medium</div>	<div><div></div>Yes</div>	Non-Standard Event Emission
UFR-01M	<div><div></div>Informational</div>	<div><div></div>Acknowledged</div>	Inefficient Redemption Fulfilment Structure

UVT-01M	<div><div></div>Medium</div>	<div><div></div>Yes</div>	Insecure Integration of Parent Dependency
UVO-01M	<div><div></div>Minor</div>	<div><div></div>Yes</div>	Inefficient & Error-Prone Vesting System
UVR-01M	<div><div></div>Minor</div>	<div><div></div>Yes</div>	Inexistent Restriction of Decimals
VPM-01M	<div><div></div>Medium</div>	<div><div></div>Acknowledged</div>	Potentially Incorrect Restriction of Price Movements

Code Style

During the manual portion of the audit, we identified **20 optimizations** that can be applied to the codebase that will decrease the operational cost associated with the execution of a particular function and generally ensure that the project complies with the latest best practices and standards in Solidity.

Additionally, this section of the audit contains any opinionated adjustments we believe the code should make to make it more legible as well as truer to its purpose.

These optimizations are enumerated below:

ID	Severity	Addressed	Title
AUS-01C	<div><div></div>Informational</div>	<div><div></div>Yes</div>	Structure Tight-Packing
BCA-01C	<div><div></div>Informational</div>	<div><div></div>Yes</div>	Non-Standard Usage of Library
BCA-02C	<div><div></div>Informational</div>	<div><div></div>Yes</div>	Redundant External Self-Calls
UFR-01C	<div><div></div>Informational</div>	<div><div></div>Yes</div>	Non-Standard Usage of Library
UVT-01C	<div><div></div>Informational</div>	<div><div></div>Yes</div>	Illegible Numeric Value Representations
UVT-02C	<div><div></div>Informational</div>	<div><div></div>Yes</div>	Non-Standard Usage of Library
UVT-03C	<div><div></div>Informational</div>	<div><div></div>Yes</div>	Repetitive Value Literal
UVO-01C	<div><div></div>Informational</div>	<div><div></div>Yes</div>	Calculation Simplification
UVO-02C	<div><div></div>Informational</div>	<div><div></div>Yes</div>	Ineffective Boolean Flag Management
UVO-03C	<div><div></div>Informational</div>	<div><div></div>Yes</div>	Ineffectual Usage of Safe Arithmetics

UVO-04C	Informational	Yes	Redundant Precision Concept
UVO-05C	Informational	Yes	Repetitive Value Literal
UVR-01C	Informational	Yes	Ineffectual Usage of Safe Arithmetics
UVR-02C	Informational	Yes	Inexecutable Code
UVR-03C	Informational	Yes	Optimization of Yielded Statement
UVR-04C	Informational	Yes	Repetitive Invocations of Storage Offset
VPM-01C	Informational	Yes	Inefficient Event Argument
VPM-02C	Informational	Nullified	Inefficient Imposition of Access Control
VPM-03C	Informational	Yes	Inefficient mapping Lookups
VPM-04C	Informational	Yes	Repetitive Value Literal

BaseControlledAsyncRedeem Static Analysis Findings

BCA-01S: Inexistent Sanitization of Input Address

Type	Severity	Location
Input Sanitization	<div><div></div>Informational</div>	BaseControlledAsyncRedeem.sol:L83-L101

Description:

The linked function accepts an `address` argument yet does not properly sanitize it.

Impact:

The presence of zero-value addresses, especially in `constructor` implementations, can cause the contract to be permanently inoperable. These checks are advised as zero-value inputs are a common side-effect of off-chain software related bugs.

Example:

src/vaults/BaseControlledAsyncRedeem.sol

SOL

```
83 function initialize(
84     BaseControlledAsyncRedeemInitParams memory params
85 ) public virtual onlyInitializing {
86     require(params.asset != address(0), ZeroAssetAddress());
87
88     // Init OZ contracts
89     __AccessControl_init();
90     __AccessControlDefaultAdminRules_init(0, params.owner);
91     __Pausable_init();
92     __ERC20_init(params.name, params.symbol);
```

Example (Cont.):

SOL

```
93     __ERC4626_init(IERC20(params.asset));
94
95     // Init self
96     _getBaseAsyncRedeemStorage().rateProvider =
97     IUltraVaultRateProvider(params.rateProvider);
98
99     // Grant roles to owner
100    _grantRole(OPERATOR_ROLE, params.owner);
101    _grantRole(PAUSER_ROLE, params.owner);
102 }
```


Recommendation:

We advise some basic sanitization to be put in place by ensuring that the `address` specified is non-zero.

Alleviation (28f27853965de07fb79f4f2b5fed696d35120032):

The input `params.rateProvider` address argument of the

`BaseControlledAsyncRedeem::initialize` function is adequately sanitized as non-zero in the latest in-scope revision of the codebase, addressing this exhibit.

OracleAdmin Static Analysis Findings

OAN-01S: Inexistent Sanitization of Input Address

Type	Severity	Location
Input Sanitization	<div><div></div>Informational</div>	OracleAdmin.sol:L43-L45

Description:

The linked function accepts an `address` argument yet does not properly sanitize it.

Impact:

The presence of zero-value addresses, especially in `constructor` implementations, can cause the contract to be permanently inoperable. These checks are advised as zero-value inputs are a common side-effect of off-chain software related bugs.

Example:

```
src/oracles/OracleAdmin.sol
SOL
43 constructor(address _oracle, address _owner) Ownable(_owner) {
44     oracle = IUltraVaultOracle(_oracle);
45 }
```

Recommendation:

We advise some basic sanitization to be put in place by ensuring that the `address` specified is non-zero.

Alleviation (28f27853965de07fb79f4f2b5fed696d35120032):

The input `_oracle` address argument of the `OracleAdmin::constructor` function is adequately sanitized as non-zero in the latest in-scope revision of the codebase, addressing this exhibit.

RedeemQueue Static Analysis Findings

RQE-01S: Inexistent Visibility Specifier

Type	Severity	Location
Code Style	<div><div></div>Informational</div>	RedeemQueue.sol:L20

Description:

The linked variable has no visibility specifier explicitly set.

Example:

src/vaults/accounting/RedeemQueue.sol

SOL

20 uint256 constant UINT128_MAX = type(uint128).max;

Recommendation:

We advise one to be set so to avoid potential compilation discrepancies in the future as the current behaviour is for the compiler to assign one automatically which may deviate between `pragma` versions.

Alleviation (28f27853965de07fb79f4f2b5fed696d35120032):

The `internal` visibility specifier has been introduced to the referenced variable, preventing potential compilation discrepancies and addressing this exhibit.

BaseControlledAsyncRedeem Manual Review Findings

BCA-01M: Incompatibility of System with EIP-777 Assets

Type	Severity	Location
Logical Fault	<div><div></div>Informational</div>	BaseControlledAsyncRedeem.sol:L388

Description:

The `BaseControlledAsyncRedeem` contract is not compatible with **EIP-777** style tokens as the before and after hooks invoked during several functions (i.e. `BaseControlledAsyncRedeem::_performDeposit`) will be vulnerable to re-entrancy attacks.

Example:

```
src/vaults/BaseControlledAsyncRedeem.sol

SOL

374 /// @dev Internal function to process deposit and mint flows
375 function _performDeposit(
376     address _asset,
377     uint256 assets,
378     uint256 shares,
379     address receiver
380 ) internal {
381     // Checks
382     require(assets != 0, EmptyDeposit());
383     require(shares != 0, NothingToMint());
```

Example (Cont.):

SOL

```
384
385 // Pre-deposit hook - use the actual asset amount being transferred
386 beforeDeposit(_asset, assets, shares);
387
388 // Need to transfer before minting or ERC777s could reenter
389 SafeERC20.safeTransferFrom(
390     IERC20(_asset),
391     msg.sender,
392     address(this),
393     assets
394 );
395
396 // Mint shares to receiver
397 _mint(receiver, shares);
398
399 // Emit event
400 emit Deposit(msg.sender, receiver, assets, shares);
401
402 // After-deposit hook - use the actual asset amount that was transferred
403 afterDeposit(_asset, assets, shares);
404 }
```

Recommendation:

We advise this notice to be omitted and support for non-standard **EIP-20** tokens to be eliminated from the system.

Alternatively, we advise re-entrancy guards to be introduced throughout all core functions of the system as well as its derivative implementations to avoid re-entrancy complications.

Alleviation (28f2785396):


The Ultra Yield team stated that they do not wish to support **EIP-777** assets yet comments remain in the code indicating otherwise.

For this item to be considered addressed, we require any comments alluding to **EIP-777** support to be omitted.

Alleviation (2d9651dbd7):

Any mention of **EIP-777** support has been omitted, addressing this exhibit.

BCA-02M: Improper Accounting Error Drift

Type	Severity	Location
Mathematical Operations	 Minor	BaseControlledAsyncRedeem.sol:L461, L521,

Description:

The `BaseControlledAsyncRedeem::_performWithdraw` function is invoked with the results of share-to-asset and asset-to-share conversions that round in favour of the protocol (i.e. asset-to-share rounds upward, share-to-asset rounds downward).

Due to this approach, it is possible for a user to have a `ClaimableRedeemSlot` data entry with a non-zero amount of assets and a zero amount of shares rendering them permanently irredeemable.

Impact:

It is presently possible for truncation errors to result in an irredeemable balance for a user.

Example:

src/vaults/BaseControlledAsyncRedeem.sol

```
SOL
510 /// @dev Internal function for redeeming exact number of `shares` and withdrawing
511 /// @dev assets in `_asset` to `receiver`
512 function _redeemAsset(
513     address _asset,
514     uint256 shares,
515     address receiver,
516     address controller
517 ) internal checkAccess(controller) returns (uint256 assets) {
518     require(shares != 0, NothingToRedeem());
519 }
```

Example (Cont.):

SOL

```
520    // Calculate assets directly in asset units
521    assets = _calculateClaimableAssetsForShares(controller, _asset, shares);
522    require(assets != 0, NothingToWithdraw());
523
524    // Execute withdraw
525    _performWithdraw(_asset, assets, shares, receiver, controller);
526 }
```

Recommendation:


We advise the code to permit a full withdrawal of the remaining assets if `0` shares remain in a `ClaimableRedeemSlot` data entry, ensuring no funds can be lost due to truncations during partial withdrawals.

Alleviation (28f27853965de07fb79f4f2b5fed696d35120032):

The Ultra Yield team evaluated the behaviour described and consider it to be in-line with conventional **EIP-4626** vault functionality that rounds in favour of the system.

As such, we consider this exhibit safely acknowledged.

BCA-03M: Incorrect Event Emission

Type	Severity	Location
Logical Fault	 Minor	BaseControlledAsyncRedeem.sol:L1016

Description:

The referenced event emission will yield the mutated `$.rateProvider` variable incorrectly.

Impact:

The data emitted through the `RateProviderUpdated` is incorrect.

Example:

src/vaults/BaseControlledAsyncRedeem.sol

```
SOL

100
2  /// @notice Accept proposed rate provider
100
3  /// @dev Pauses vault to ensure provider setup and prevent deposits with faulty
prices
100
4  /// @dev Oracle must be switched before unpausing
100
5  function acceptProposedRateProvider(address newRateProvider) external onlyOwner
{
100
6      BaseAsyncRedeemStorage storage $ = _getBaseAsyncRedeemStorage();
100
7      AddressUpdateProposal memory proposal = $.proposedRateProvider;
100
8
100
9      require(proposal.addr != address(0), NoRateProviderProposed());
101
0      require(proposal.addr == newRateProvider, ProposedRateProviderMismatch());
101
1      require(block.timestamp >= proposal.timestamp + ADDRESS_UPDATE_TIMELOCK,
CannotAcceptRateProviderYet());
```

Example (Cont.):

SOL

```
101
2     require(block.timestamp <= proposal.timestamp + MAX_ADDRESS_UPDATE_WAIT,
RateProviderUpdateExpired());
101
3
101
4     $.rateProvider = IUltraVaultRateProvider(proposal.addr);
101
5     delete $.proposedRateProvider;
101
6     emit RateProviderUpdated(address($.rateProvider), proposal.addr);
101
7
101
8     // Pause to manually check the setup by operators
101
9     _pause();
102
0 }
```


Recommendation:

We advise the event's emission to be relocated prior to the `$.rateProvider` assignment, ensuring the data yielded by the `RateProviderUpdated` event is correct.

Alleviation (28f27853965de07fb79f4f2b5fed696d35120032):

The event's emission was corrected, ensuring that the proper rate provider data points are emitted.

BCA-04M: Non-Standard Event Emission

Type	Severity	Location
Logical Fault	 Medium	BaseControlledAsyncRedeem.sol:L806

Description:

The **EIP-7540** standard mandates that the `RedeemRequest` event is emitted during redemption requests, however, the system uses a custom `RedeemRequested` event instead.

Impact:

Off-chain software meant to integrate with **EIP-7540** vaults will fail to do so properly due to absence of **MUST** keyword events defined in the **EIP-7540** standard.

Example:

src/vaults/BaseControlledAsyncRedeem.sol

SOL

```
782 /// @dev Internal function for processing redeem requests
783 function _requestRedeemOfAsset(
784     address _asset,
785     uint256 shares,
786     address controller,
787     address owner
788 ) internal checkAccess(owner) returns (uint256 requestId) {
789     // Checks
790     require(shares != 0, NothingToRedeem());
791     require(IERC20(address(this)).balanceOf(owner) >= shares,
792     InsufficientBalance());
```

Example (Cont.):

S0L

```
792
793     // Validate that the asset is supported by the rate provider
794     require(_asset == this.asset() || rateProvider().isSupported(_asset),
AssetNotSupported());
795
796     // Call beforeRequestRedeem hook
797     beforeRequestRedeem(_asset, shares, controller, owner);
798
799     // Update pending redeem
800     _increasePendingRedeem(controller, _asset, shares);
801
802     // Transfer shares to vault for burning later
803     SafeERC20.safeTransferFrom(IERC20(this), owner, address(this), shares);
804
805     // Emit event
806     emit RedeemRequested(controller, owner, REQUEST_ID, msg.sender, shares);
807
808     return REQUEST_ID;
809 }
```


Recommendation:

We advise the proper event signature to be satisfied so as to be compliant with the **EIP-7540** standard.

Alleviation (28f27853965de07fb79f4f2b5fed696d35120032):

The **event** was aptly renamed to ensure it complies with the **EIP-7540** standard, alleviating this exhibit.

UltraFeeder Manual Review Findings

UFR-01M: Inefficient Redemption Fulfilment Structure

Type	Severity	Location
Gas Optimization	<div><div></div>Informational</div>	UltraFeeder.sol:L186-L195, L198-L208, L212-L222

Description:

The referenced redemption fulfilment function structure is complicated unnecessarily, relying on before, after, as well as extra after-execution logic for redemption fulfilments.

Impact:

While this is a gas optimization, the complexity of the code is prone to error and must be simplified.

Example:

src/vaults/UltraFeeder.sol

SOL

```
184 /// @dev Before fulfill redeem - transfer funds from fundsHolder to vault
185 /// @dev "assets" will already be correct given the token user requested
186 function beforeFulfillRedeem(address _asset, uint256 assets, uint256 shares)
internal override {
187     IUltraVault mainVault_ = mainVault();
188     // Fulfill redeem in main vault. Returns asset units
189     mainVault_.fulfillRedeemOfAsset(_asset, shares, address(this));
190     uint256 mainAssetsClaimed = mainVault_.redeemAsset(_asset, shares,
address(this), address(this));
191
192     // Deduct the expected withdrawal fee from the total amount of assets
193     uint256 expectedAssetsAfterFees = assets -
mainVault_.calculateWithdrawalFee(assets);
```

Example (Cont.):

S0L

```
194     require(mainAssetsClaimed == expectedAssetsAfterFees, AssetNumberMismatch());
195 }
196
197 /// @dev Internal fulfill redeem request logic
198 function _fulfillRedeemOfAsset(
199     address _asset,
200     uint256 assets,
201     uint256 shares,
202     address controller
203 ) internal override returns (uint256) {
204     uint256 assetsFulfilled = super._fulfillRedeemOfAsset(_asset, assets, shares,
205 controller);
206     // Deduct the expected withdrawal fee from the total amount of assets
207     uint256 withdrawalFee = mainVault().calculateWithdrawalFee(assetsFulfilled);
208     return assetsFulfilled - withdrawalFee;
209 }
210
211 /// @dev Hook for inheriting contracts after fulfill redeem
212 /// @dev Correct claimable redeem amounts to account for underlying vault fees
213 function afterFulfillRedeem(
214     address _asset,
215     uint256 assets,
216     uint256, // shares
217     address controller
218 ) internal override {
219     // The base implementation has added `_assets` to the claimable assets
220     // We need to calculate the withdrawal fee and deduct it from the claimable
221     assets
222     uint256 withdrawalFee = mainVault().calculateWithdrawalFee(assets);
223     _consumeClaimableRedeem(controller, _asset, withdrawalFee, 0);
224 }
```

Example (Cont.):

SOL

222 }

Recommendation:


We advise the code of the `BaseControlledAsyncRedeem::_fulfillRedeemOfAsset` function to be updated to utilize a return value by the `UltraFeeder::beforeFulfillRedeem` function, permitting the actual assets to be redeemed to be yielded and thus simplifying the code greatly.

Alleviation (28f27853965de07fb79f4f2b5fed696d35120032):

The Ultra Yield team evaluated this exhibit and opted to not implement the optimization outlined as it would significantly affect the logic of the `UltraVault` and `UltraFeeder` contracts.

UltraVault Manual Review Findings

UVT-01M: Insecure Integration of Parent Dependency

Type	Severity	Location
Logical Fault	 Medium	UltraVault.sol:L171-L173

Description:

The `UltraVault::beforeDeposit` function is meant to integrate into the pre-deposit hook of the parent `BaseControlledAsyncRedeem::_performDeposit` function, however, in doing so it will cause all incoming deposits to be immediately diluted.

Specifically, the `BaseControlledAsyncRedeem::_performDeposit` function is invoked with an `assets` and `shares` pair reflecting the state of the vault **before the fees have been captured**.

As the fees result in an increase of the `totalSupply` of the token, the actual share-to-asset ratio is mutated within the `UltraVault::beforeDeposit` function even though the original `BaseControlledAsyncRedeem::_performDeposit` function would continue with the pre-fee share-to-asset ratio.

This results in a consistent over-dilution of new depositors which is incorrect.

Impact:

All new depositors are unfairly diluted as they will receive less shares than their assets would normally entail, causing existing shareholders to receive the diluted deposit's proceeds.

Example:

src/vaults/UltraVault.sol

SOL

```
169 /// @notice Collect fees before deposit
170 /// @dev Expected to be overridden in inheriting contracts
171 function beforeDeposit(address, uint256, uint256) internal virtual override {
172     _collectFees();
173 }
```

Recommendation:

We advise fee collection to occur through overridden

`BaseControlledAsyncRedeem::_mintWithAsset` and `BaseControlledAsyncRedeem::_depositAsset` functions, ensuring fees are captured prior to the preview functions and thus prior to the total supply measurements that deposit operations would rely on.

Alleviation (28f27853965de07fb79f4f2b5fed696d35120032):

The code was updated per our recommendation, overriding the relevant

`BaseControlledAsyncRedeem` functions to ensure that fees are imposed correctly and securely.

UltraVaultOracle Manual Review Findings

UVO-01M: Inefficient & Error-Prone Vesting System

Type	Severity	Location
Logical Fault	<div><div></div>Minor</div>	UltraVaultOracle.sol:L151, L155, L205, L206, L207

Description:

The vesting system in use by the `UltraVaultOracle::_getCurrentPrice` function is ineffective and results in rounding errors that accelerate price movements.

Specifically, the price change is evaluated in the inverse direction (i.e. a delta of the full price change based on the time that has elapsed) which will result in abrupt price changes if a truncation occurs.

Impact:

It is possible for a price vesting operation to achieve the `targetPrice` before the timestamp for full vesting has been achieved due to how truncations accelerate price movements in the current implementation.

Example:

src/oracles/UltraVaultOracle.sol

SOL

```
205 uint256 timeLeft = price.timestampForFullVesting - block.timestamp;
206 uint256 timeFull = price.timestampForFullVesting - price.lastUpdatedTimestamp;
207 uint256 change = diff - diff * timeLeft * DECIMAL_PRECISION / timeFull /
DECIMAL_PRECISION;
```


Recommendation:

We advise truncations to slow down the price movements instead of accelerating them by revising the system to use a time-elapsed approach.


In detail, the price `change` should be evaluated as `diff * timeElapsed / totalTime` ensuring that any downward rounding operations will not result in abrupt price movements.

Alleviation (28f27853965de07fb79f4f2b5fed696d35120032):

The code was updated by implementing a time-based approach to the way the price moves between the `price` and `targetPrice` data points, alleviating this exhibit.

UltraVaultRateProvider Manual Review Findings

UVR-01M: Inexistent Restriction of Decimals

Type	Severity	Location
Mathematical Operations	 Minor	UltraVaultRateProvider.sol:L102

Description:

The decimals of an asset are not restricted when dealing with a pegged asset, an approach that may result in an overflow or underflow when converting between decimal denominations through the `UltraVaultRateProvider::_convertDecimals` function.

Impact:

Certain assets might result in exponential overflows if their decimal differs greatly from that of the base asset's.

Example:

src/oracles/UltraVaultRateProvider.sol

SOL

```
88  /// @inheritdoc IUltraVaultRateProvider
89  function addAsset(address asset, bool isPegged, address rateProvider) external
    onlyOwner {
90      // Checks
91      AssetData memory data = supportedAssets(asset);
92      require(!data.isPegged && data.rateProvider == address(0),
        AssetAlreadySupported());
93      if (isPegged) {
94          require(rateProvider == address(0), InvalidRateProvider());
95      } else {
96          require(rateProvider != address(0), InvalidRateProvider());
97      }
```

Example (Cont.):

SOL

```
98
99     // Update storage
100     _getStorage().supportedAssets[asset] = AssetData({
101         isPegged: isPegged,
102         decimals: IERC20Metadata(asset).decimals(),
103         rateProvider: rateProvider
104     });
105
106     // Emit events
107     emit AssetAdded(address(asset), isPegged);
108     if (!isPegged) {
109         emit RateProviderUpdated(address(asset), rateProvider);
110     }
111 }
```

Recommendation:

We advise a restriction to be imposed on the maximum delta between the base asset and the introduced asset, ensuring that the power operations do not result in an overflow that prohibit conversions from being properly calculated.

Alleviation (28f27853965de07fb79f4f2b5fed696d35120032):

The code was revised to always sanitize the `_decimals` of an `asset` being included in the system, ensuring they comply by they are within the `[6,18]` value range and thus do not result in an abnormal exponent.

VaultPriceManager Manual Review Findings

VPM-01M: Potentially Incorrect Restriction of Price Movements

Type	Severity	Location
Logical Fault	<div><div></div>Medium</div>	VaultPriceManager.sol:L156

Description:

The referenced statement will restrict time-based linear price updates through the `VaultPriceManager::_checkSuddenMovements` function meant for abrupt price changes.

Impact:

A linear price update that occurs over a significant amount of time may be erroneously considered "abrupt" in the current `VaultPriceManager` implementation due to applying the same sudden movement restriction across instantaneous and linear vested price updates.

Example:

```
src/oracles/VaultPriceManager.sol

SOL

151 /// @notice Internal gradual price update function
152 function _updatePriceWithVesting(
153     PriceUpdate calldata priceUpdate,
154     uint256 timestampForFullVesting
155 ) internal onlyAdminOrOwner(priceUpdate.vault) {
156     _checkSuddenMovements(priceUpdate);
157     oracle.scheduleLinearPriceUpdate(
158         priceUpdate.vault,
159         priceUpdate.asset,
160         priceUpdate.shareValueInAssets,
```

Example (Cont.):

SOL

```
161     timestampForFullVesting
162 );
163 }
164
165 /// @notice Check price update for sudden price swings and update highwatermark
166 function _checkSuddenMovements(
167     PriceUpdate calldata priceUpdate
168 ) internal {
169     uint256 lastPrice = oracle.getCurrentPrice(
170         priceUpdate.vault,
171         priceUpdate.asset
172     );
173     uint256 highwaterMark = highwaterMarks[priceUpdate.vault];
174     Limit memory limit = limits[priceUpdate.vault];
175     if (
176         // Sudden drop
177         priceUpdate.shareValueInAssets <
178         lastPrice.mulDivDown(1e18 - limit.jump, 1e18) ||
179         // Sudden increase
180         priceUpdate.shareValueInAssets >
181         lastPrice.mulDivDown(1e18 + limit.jump, 1e18) ||
182         // Drawdown check
183         priceUpdate.shareValueInAssets <
184         highwaterMark.mulDivDown(1e18 - limit.drawdown, 1e18)
185     ) {
186         IPausable vault = IPausable(priceUpdate.vault);
187         if (!vault.paused()) {
188             vault.pause();
189         }
190     }
191 }
```

Example (Cont.):

SOL

```
189     }
190   } else if (priceUpdate.shareValueInAssets > highwaterMark) {
191     highwaterMarks[priceUpdate.vault] = priceUpdate.shareValueInAssets;
192   }
193 }
```

Recommendation:

We advise a distinct restriction to be imposed on linear price updates that will take into account the time it takes for price updates to occur.

Alleviation (28f27853965de07fb79f4f2b5fed696d35120032):

The Ultra Yield team evaluated this exhibit and opted to acknowledge it as they consider vesting durations to be at most **7 days**, meaning that the same sudden price movement principle can be applied to both instantaneous changes as well as gradual ones.

By definition, these two approaches can never be identical so imposing the same limitation is a matter of usability over functionality and thus consider this exhibit to be acknowledged by the Ultra Yield team.

AddressUpdates Code Style Findings

AUS-01C: Structure Tight-Packing

Type	Severity	Location
Gas Optimization	<div><div></div>Informational</div>	AddressUpdates.sol:L5, L6

Description:

The `AddressUpdateProposal` structure can be tightly packed into a single 256-bit slot, reducing gas costs by a significant amount when dealing with data entries such as the `BaseControlledAsyncRedeem` contract's `BaseAsyncRedeemStorage` data entry.

Example:

src/utils/AddressUpdates.sol

SOL

```
4 struct AddressUpdateProposal {
5     address addr;
6     uint256 timestamp;
7 }
```

Recommendation:

We advise this to be done so by reducing the `timestamp` variable to a `uint96` data type, ensuring that it can be tightly packed with the `address` variable that precedes it.

Alleviation (28f27853965de07fb79f4f2b5fed696d35120032):

The referenced data type was properly downcast to `uint96`, optimizing the code's storage gas cost significantly.

BaseControlledAsyncRedeem Code Style Findings

BCA-01C: Non-Standard Usage of Library

Type	Severity	Location
Code Style	<div><div></div>Informational</div>	BaseControlledAsyncRedeem.sol: <ul style="list-style-type: none">• I-1: L389-L394• I-2: L543• I-3: L803• I-4: L886

Description:

The referenced statements will utilize the `SafeERC20` library via direct invocations rather than applying the library to the `IERC20` data type and utilizing it through it.

Example:

```
src/vaults/BaseControlledAsyncRedeem.sol

SOL

388 // Need to transfer before minting or ERC777s could reenter
389 SafeERC20.safeTransferFrom(
390     IERC20(_asset),
391     msg.sender,
392     address(this),
393     assets
394 );
```

Recommendation:

We advise the `using SafeERC20 for IERC20` statement to be introduced to the codebase and the relevant functions to be invoked via the `IERC20` data type, standardizing the code's syntax.

Alleviation (28f27853965de07fb79f4f2b5fed696d35120032):

All referenced library invocation instances have been properly replaced by invocations of the relevant functions directly through the data types involved, addressing this exhibit.

BCA-02C: Redundant External Self-Calls

Type	Severity	Location
Gas Optimization	<div><div></div>Informational</div>	BaseControlledAsyncRedeem.sol: <ul style="list-style-type: none">• l-1: L791• l-2: L794• l-3: L803• l-4: L886

Description:

The referenced statements will perform external self-calls which are inefficient and redundant.

Example:

src/vaults/BaseControlledAsyncRedeem.sol

SOL

791 require(IERC20(address(this)).balanceOf(owner) >= shares, InsufficientBalance());

Recommendation:

We advise the relevant functions to be invoked directly internally, optimizing their gas cost.

Alleviation (28f2785396):

While the exhibit has been partially addressed, instances **3** and **4** remain unaddressed as they continue to perform an external self-call.

Alleviation (2d9651dbd7):

The remaining two self-call instances have been optimized per our recommendation, addressing this exhibit in full.

UltraFeeder Code Style Findings

UFR-01C: Non-Standard Usage of Library

Type	Severity	Location
Code Style	<div><div></div>Informational</div>	UltraFeeder.sol: <ul style="list-style-type: none">• I-1: L163• I-2: L178

Description:

The referenced statements will utilize the `SafeERC20` library via direct invocations rather than applying the library to the `IERC20` data type and utilizing it through it.

Example:

src/vaults/UltraFeeder.sol

SOL

163 SafeERC20.safeIncreaseAllowance(IERC20(_asset), address(_mainVault), assets);

Recommendation:

We advise the `using SafeERC20 for IERC20` statement to be introduced to the codebase and the relevant functions to be invoked via the `IERC20` data type, standardizing the code's syntax.

Alleviation (28f27853965de07fb79f4f2b5fed696d35120032):

All referenced library invocation instances have been properly replaced by invocations of the relevant functions directly through the data types involved, addressing this exhibit.

UltraVault Code Style Findings

UVT-01C: Illegible Numeric Value Representations

Type	Severity	Location
Code Style	<div><div></div>Informational</div>	UltraVault.sol: <ul style="list-style-type: none">l-1: L48l-2: L49l-3: L50

Description:

The linked representations of numeric literals are sub-optimally represented decreasing the legibility of the codebase.

Example:

src/vaults/UltraVault.sol

SOL

48 uint64 public constant MAX_PERFORMANCE_FEE = 3e17; // 30%

Recommendation:

To properly illustrate each value's purpose, we advise the following guidelines to be followed. For values meant to depict fractions with a base of `1e18`, we advise fractions to be utilized directly (i.e. `1e17` becomes `0.1e18`) as they are supported. For values meant to represent a percentage base, we advise each value to utilize the underscore (`_`) separator to discern the percentage decimal (i.e. `10000` becomes `100_00`, `300` becomes `3_00` and so on). Finally, for large numeric values we simply advise the underscore character to be utilized again to represent them (i.e. `1000000` becomes `1_000_000`).

Alleviation (28f27853965de07fb79f4f2b5fed696d35120032):

The referenced literal representations have been corrected by utilizing multiples of a `ONE_PERCENT` literal, addressing this exhibit.

UVT-02C: Non-Standard Usage of Library

Type	Severity	Location
Code Style	<div><div></div>Informational</div>	UltraVault.sol: <ul style="list-style-type: none">• I-1: L178• I-2: L184• I-3: L553

Description:

The referenced statements will utilize the `SafeERC20` library via direct invocations rather than applying the library to the `IERC20` data type and utilizing it through it.

Example:

src/vaults/UltraVault.sol

SOL

553 SafeERC20.safeTransferFrom(IERC20(_asset), fundsHolder(), feeRecipient(), fee);

Recommendation:

We advise the `using SafeERC20 for IERC20` statement to be introduced to the codebase and the relevant functions to be invoked via the `IERC20` data type, standardizing the code's syntax.

Alleviation (28f27853965de07fb79f4f2b5fed696d35120032):

All referenced library invocation instances have been properly replaced by invocations of the relevant functions directly through the data types involved, addressing this exhibit.

UVT-03C: Repetitive Value Literal

Type	Severity	Location
Code Style	<div><div></div>Informational</div>	UltraVault.sol:L207, L239, L279, L412, L541

Description:

The linked value literal is repeated across the codebase multiple times.

Example:

src/vaults/UltraVault.sol

SOL

207 uint256 withdrawalFee = assets.mulDivDown(getFees().withdrawalFee, 1e18);

Recommendation:

We advise it to be set to a `constant` variable instead, optimizing the legibility of the codebase.

In case the `constant` has already been declared, we advise it to be properly re-used across the code.

Alleviation (28f27853965de07fb79f4f2b5fed696d35120032):

The referenced value literal `1e18` has been properly relocated to a contract-level `constant` declaration labelled `ONE_UNIT`, optimizing the code's legibility.

UltraVaultOracle Code Style Findings

UVO-01C: Calculation Simplification

Type	Severity	Location
Gas Optimization	<div><div></div>Informational</div>	UltraVaultOracle.sol:L233

Description:

The referenced calculation can be simplified by merging exponentials into a single multiplication or division depending on the outcome.

Example:

src/oracles/UltraVaultOracle.sol

SOL

233 return inAmount * price * (10 ** quoteDecimals) / (10 ** (baseDecimals + 18));

Recommendation:

We advise the powers to be summed and to perform a single multiplication or division with the relevant power of `10`.

Alleviation (28f27853965de07fb79f4f2b5fed696d35120032):

The code was updated per our recommendation albeit ensuring that the `denominatorDecimals` are always greater than the `nominatorDecimals`, simplifying the calculation further.

UVO-02C: Ineffective Boolean Flag Management

Type	Severity	Location
Gas Optimization	<div><div></div>Informational</div>	UltraVaultOracle.sol:L195, L198, L199

Description:

The `increase` boolean is meant to indicate whether `price.price <= price.targetPrice` is true.

Example:

```
src/oracles/UltraVaultOracle.sol
SOL
195 bool increase;
196 uint256 diff;
197
198 if (price.price <= price.targetPrice) {
199     increase = true;
200     diff = price.targetPrice - price.price;
201 } else {
202     diff = price.price - price.targetPrice;
203 }
```

Recommendation:

We advise the `increase` flag to be assigned to the `price.price <= price.targetPrice` evaluation on its declaration and the `if` conditional to utilize the `increase` flag directly, optimizing the code's gas cost.

Alleviation (28f27853965de07fb79f4f2b5fed696d35120032):

The `increase` flag's assignment has been revised as advised, optimizing the code's gas cost.

UVO-03C: Ineffectual Usage of Safe Arithmetics

Type	Severity	Location
Language Specific	<div><div></div>Informational</div>	UltraVaultOracle.sol: <ul style="list-style-type: none">l-1: L200l-2: L202

Description:

The linked mathematical operations are guaranteed to be performed safely by surrounding conditionals evaluated in either `require` checks or `if-else` constructs.

Example:

src/oracles/UltraVaultOracle.sol

SOL

```
198 if (price.price <= price.targetPrice) {
199     increase = true;
200     diff = price.targetPrice - price.price;
201 } else {
202     diff = price.price - price.targetPrice;
203 }
```

Recommendation:

Given that safe arithmetics are toggled on by default in `pragma` versions of `0.8.X`, we advise the linked statements to be wrapped in `unchecked` code blocks thereby optimizing their execution cost.

Alleviation (28f27853965de07fb79f4f2b5fed696d35120032):

The referenced price delta calculations are now wrapped in an `unchecked` code block, optimizing the code's gas cost.

UVO-04C: Redundant Precision Concept

Type	Severity	Location
Gas Optimization	<div><div></div>Informational</div>	UltraVaultOracle.sol:L21, L207

Description:

The `DECIMAL_PRECISION` concept employed by the `UltraVaultOracle::_getCurrentPrice` function is ineffective as the extrapolation of the `timeLeft` is immediately offset after use, thereby resulting in no increase of the calculation's accuracy.

Example:

src/oracles/UltraVaultOracle.sol

SOL

207 uint256 change = diff - diff * timeLeft * DECIMAL_PRECISION / timeFull /
DECIMAL_PRECISION;

Recommendation:

We advise the concept to be entirely omitted, optimizing the code's gas cost.

While precision errors exist, they are detailed in a separate exhibit.

Alleviation (28f27853965de07fb79f4f2b5fed696d35120032):

The ineffective precision concept has been omitted from the code, optimizing its legibility and gas cost.

UVO-05C: Repetitive Value Literal

Type	Severity	Location
Code Style	<div><div></div>Informational</div>	UltraVaultOracle.sol:L233, L252

Description:

The linked value literal is repeated across the codebase multiple times.

Example:

src/oracles/UltraVaultOracle.sol

SOL

233 return inAmount * price * (10 ** quoteDecimals) / (10 ** (baseDecimals + 18));

Recommendation:

We advise it to be set to a `constant` variable instead, optimizing the legibility of the codebase.

In case the `constant` has already been declared, we advise it to be properly re-used across the code.

Alleviation (28f27853965de07fb79f4f2b5fed696d35120032):

The referenced value literal `18` has been properly relocated to a contract-level `constant` declaration labelled `PRICE_FEED_DECIMALS`, optimizing the code's legibility.

UltraVaultRateProvider Code Style Findings

UVR-01C: Ineffectual Usage of Safe Arithmetics

Type	Severity	Location
Language Specific	<div><div></div>Informational</div>	UltraVaultRateProvider.sol: <ul style="list-style-type: none">• l-1: L179• l-2: L181

Description:

The linked mathematical operation is guaranteed to be performed safely by logical inference, such as surrounding conditionals evaluated in `require` checks or `if-else` constructs.

Example:

src/oracles/UltraVaultRateProvider.sol

SOL

```
178 } else if (fromDecimals < toDecimals) {
179     return amount * 10 ** (toDecimals - fromDecimals);
180 } else {
181     return amount / 10 ** (fromDecimals - toDecimals);
182 }
```


Recommendation:

Given that safe arithmetics are toggled on by default in `pragma` versions of `0.8.X`, we advise the linked statement to be wrapped in an `unchecked` code block thereby optimizing its execution cost.

Alleviation (28f27853965de07fb79f4f2b5fed696d35120032):

Both decimal difference calculations have been wrapped in an `unchecked` code block, optimizing the code's gas cost.

UVR-02C: Inexecutable Code

Type	Severity	Location
Gas Optimization	 Informational	UltraVaultRateProvider.sol:L176-L178

Description:

The first case of the referenced `if-else-if` conditional chain will never be executed as the `UltraVaultRateProvider::_convertDecimals` function is solely invoked whenever the decimals differ between them.

Example:

src/oracles/UltraVaultRateProvider.sol

SOL

```
170 /// @dev Helps with decimals accounting
171 function _convertDecimals(
172     uint256 amount,
173     uint8 fromDecimals,
174     uint8 toDecimals
175 ) internal pure returns (uint256) {
176     if (fromDecimals == toDecimals) {
177         return amount;
178     } else if (fromDecimals < toDecimals) {
179         return amount * 10 ** (toDecimals - fromDecimals);
```

Example (Cont.):

SOL

```
180     } else {  
181         return amount / 10 ** (fromDecimals - toDecimals);  
182     }  
183 }
```


Recommendation:

We advise the code to be optimized based on this premise, reducing its gas cost.

Alleviation (28f27853965de07fb79f4f2b5fed696d35120032):

The code was updated to instead invoke the `UltraVaultRateProvider::_convertDecimals` function in the decimal equality case as well, addressing this exhibit indirectly.

UVR-03C: Optimization of Yielded Statement

Type	Severity	Location
Gas Optimization	 Informational	UltraVaultRateProvider.sol:L82-L85

Description:

The `UltraVaultRateProvider::isSupported` function will evaluate whether `data.isPegged` is `true`, yield `true` in such a case, and yield `data.rateProvider != address(0)` in any other.

Example:

src/oracles/UltraVaultRateProvider.sol

SOL

```
82  if (data.isPegged) {  
83      return true;  
84  }  
85  return data.rateProvider != address(0);
```

Recommendation:

We advise the code to directly yield `data.isPegged || data.rateProvider != address(0)`, optimizing the code's gas cost.

Alleviation (28f27853965de07fb79f4f2b5fed696d35120032):

The code was updated to yield an optimized conditional statement, addressing this exhibit.

UVR-04C: Repetitive Invocations of Storage Offset

Type	Severity	Location
Gas Optimization	<div><div></div>Informational</div>	UltraVaultRateProvider.sol: <ul style="list-style-type: none">• I-1: L91, L100• I-2: L115, L116• I-3: L122, L123, L126• I-4: L132, L133• I-5: L150, L151

Description:

The referenced statement pairs will end up invoking the `UltraVaultRateProvider::_getStorage` function multiple times within the same code block redundantly.

Example:

```
src/oracles/UltraVaultRateProvider.sol

SOL

121 function updateRateProvider(address asset, address rateProvider) external
onlyOwner {
122     require(asset != baseAsset(), CannotUpdateBaseAsset());
123     require(!supportedAssets(asset).isPegged, AssetNotSupported());
124     require(rateProvider != address(0), InvalidRateProvider());
125
126     _getStorage().supportedAssets[asset].rateProvider = rateProvider;
127     emit RateProviderUpdated(address(asset), rateProvider);
128 }
```


Recommendation:

We advise the `UltraVaultRateProvider::_getStorage` function to be invoked once, cached to a local variable, and then re-used to optimize the code's gas cost.

Alleviation (28f2785396):

The optimization was solely applied to the example instance, rendering instances 1, 3, 4, and 5 to remain unaddressed.

Alleviation (2d9651dbd7):

All remaining instances have been optimized per our recommendation, addressing this exhibit in full.

VaultPriceManager Code Style Findings

VPM-01C: Inefficient Event Argument

Type	Severity	Location
Gas Optimization	<div><div></div>Informational</div>	VaultPriceManager.sol:L264, L269

Description:

The referenced `LimitsUpdated` event will fetch the `limits[_vault]` data entry from storage even though a local variable containing its data points is already present in memory.

Example:

src/oracles/VaultPriceManager.sol

SOL

```
260 function _setLimits(address _vault, Limit memory _limit) internal {
261     require(_limit.jump <= 1e18 && _limit.drawdown <= 1e18, InvalidLimit());
262
263
264     Limit memory oldLimit = limits[_vault];
265     if (
266         _limit.jump != oldLimit.jump ||
267         _limit.drawdown != oldLimit.drawdown
268     ) {
269         emit LimitsUpdated(_vault, limits[_vault], _limit);
270     }
```

Example (Cont.):

SOL

```
270         limits[_vault] = _limit;
271     }
272 }
```

Recommendation:

We advise the `oldLimit` variable to be emitted instead, optimizing the code's gas cost.

Alleviation (28f27853965de07fb79f4f2b5fed696d35120032):

The `event` emission was updated to utilize the local `memory` limit variable, optimizing its gas cost significantly.

VPM-02C: Inefficient Imposition of Access Control

Type	Severity	Location
Gas Optimization	<div><div></div>Informational</div>	VaultPriceManager.sol:L243, L253, L260

Description:

The `VaultPriceManager::setLimits` function implementations both impose the `Ownable2Step::onlyOwner` modifier independently even though both end up invoking the internal `VaultPriceManager::_setLimits` function.

Example:

```
src/oracles/VaultPriceManager.sol

SOL

240 /// @notice Set vault price limits
241 /// @param _vault Vault address
242 /// @param _limit Price limits to set
243 function setLimits(address _vault, Limit memory _limit) external onlyOwner {
244     _setLimits(_vault, _limit);
245 }
246
247 /// @notice Set price limits for multiple vaults
248 /// @param _vaults Array of vault addresses
249 /// @param _limits Array of price limits
```

Example (Cont.):

SOL

```
250 function setLimits(  
251     address[] memory _vaults,  
252     Limit[] memory _limits  
253 ) external onlyOwner {  
254     require(_vaults.length == _limits.length, InputLengthMismatch());  
255     for (uint256 i; i < _vaults.length; i++) {  
256         _setLimits(_vaults[i], _limits[i]);  
257     }  
258 }  
259  
260 function _setLimits(address _vault, Limit memory _limit) internal {
```

Recommendation:

We advise the `VaultPriceManager::_setLimits` function to be invoked, optimizing the code's bytecode size.

Alleviation (28f27853965de07fb79f4f2b5fed696d35120032):

The Ultra Yield team evaluated this exhibit and clarified that the `VaultPriceManager::setLimits` loop function variant would incur a higher gas cost due to imposing the access control several times in a single loop.

As such, we consider the gas benefit to outweigh the bytecode size minimization in this case and thus assess this exhibit as invalid.

VPM-03C: Inefficient mapping Lookups

Type	Severity	Location
Gas Optimization	<div><div></div>Informational</div>	VaultPriceManager.sol:L223, L225

Description:

The linked statements perform key-based lookup operations on `mapping` declarations from storage multiple times for the same key redundantly.

Example:

```
src/oracles/VaultPriceManager.sol
SOL
218 function setAdmin(
219     address _vault,
220     address _admin,
221     bool _isAdmin
222 ) external onlyOwner {
223     if (isAdmin[_vault][_admin] != _isAdmin) {
224         emit AdminUpdated(_vault, _admin, _isAdmin);
225         isAdmin[_vault][_admin] = _isAdmin;
226     }
227 }
```


Recommendation:

As the lookups internally perform an expensive `keccak256` operation, we advise the lookups to be cached wherever possible to a single local declaration that either holds the value of the `mapping` in case of primitive types or holds a `storage` pointer to the `struct` contained.

As the compiler's optimizations may take care of these caching operations automatically at-times, we advise the optimization to be selectively applied, tested, and then fully adopted to ensure that the proposed caching model indeed leads to a reduction in gas costs.

Alleviation (28f27853965de07fb79f4f2b5fed696d35120032):

The Ultra Yield team opted to acknowledge this exhibit in favour of code legibility.

VPM-04C: Repetitive Value Literal

Type	Severity	Location
Code Style	<div><div></div>Informational</div>	VaultPriceManager.sol:L178, L181, L184, L261

Description:

The linked value literal is repeated across the codebase multiple times.

Example:

```
src/oracles/VaultPriceManager.sol
SOL
178 lastPrice.mulDivDown(1e18 - limit.jump, 1e18) ||
```

Recommendation:

We advise it to be set to a `constant` variable instead, optimizing the legibility of the codebase.

In case the `constant` has already been declared, we advise it to be properly re-used across the code.

Alleviation (28f27853965de07fb79f4f2b5fed696d35120032):

The referenced value literal `1e18` has been properly relocated to a contract-level `constant` declaration labelled `SCALE`, optimizing the code's legibility.

Finding Types

A description of each finding type included in the report can be found below and is linked by each respective finding. A full list of finding types Omniscia has defined will be viewable at the central audit methodology we will publish soon.

Input Sanitization

As there are no inherent guarantees to the inputs a function accepts, a set of guards should always be in place to sanitize the values passed in to a particular function.

Indeterminate Code

These types of issues arise when a linked code segment may not behave as expected, either due to mistyped code, convoluted if blocks, overlapping functions / variable names and other ambiguous statements.

Language Specific

Language specific issues arise from certain peculiarities that the Circom language boasts that discerns it from other conventional programming languages.

Curve Specific

Circom defaults to using the BN128 scalar field (a 254-bit prime field), but it also supports BSL12-381 (which has a 255-bit scalar field) and Goldilocks (with a 64-bit scalar field). However, since there are no constants denoting either the prime or the prime size in bits available in the Circom language, some Circomlib templates like `Sign` (which returns the sign of the input signal), and `AliasCheck` (used by the strict versions of `Num2Bits` and `Bits2Num`), hardcode either the BN128 prime size or some other constant related to BN128. Using these circuits with a custom prime may thus lead to unexpected results and should be avoided.

Code Style

In these types of findings, we identify whether a project conforms to a particular naming convention and whether that convention is consistent within the codebase and legible. In case of inconsistencies, we point them out under this category. Additionally, variable shadowing falls under this category as well which is identified when a local-level variable contains the same name as a toplevel variable in the circuit.

Mathematical Operations

This category is used when a mathematical issue is identified. This implies an issue with the implementation of a calculation compared to the specifications.

Logical Fault

This category is a bit broad and is meant to cover implementations that contain flaws in the way they are implemented, either due to unimplemented functionality, unaccounted-for edge cases or similar extraordinary scenarios.

Privacy Concern

This category is used when information that is meant to be kept private is made public in some way.

Proof Concern

Under-constrained signals are one of the most common issues in zero-knowledge circuits. Issues with proof generation fall under this category.

Severity Definition

In the ever-evolving world of blockchain technology, vulnerabilities continue to take on new forms and arise as more innovative projects manifest, new blockchain-level features are introduced, and novel layer-2 solutions are launched. When performing security reviews, we are tasked with classifying the various types of vulnerabilities we identify into subcategories to better aid our readers in understanding their impact.

Within this page, we will clarify what each severity level stands for and our approach in categorizing the findings we pinpoint in our audits. To note, all severity assessments are performed **as if the contract's logic cannot be upgraded** regardless of the underlying implementation.

Severity Levels

There are five distinct severity levels within our reports; `unknown`, `informational`, `minor`, `medium`, and `major`. A TL;DR overview table can be found below as well as a dedicated chapter to each severity level:

	Impact (None)	Impact (Low)	Impact (Moderate)	Impact (High)
Likelihood (None)	<div><div></div>Informational</div>	<div><div></div>Informational</div>	<div><div></div>Informational</div>	<div><div></div>Informational</div>
Likelihood (Low)	<div><div></div>Informational</div>	<div><div></div>Minor</div>	<div><div></div>Minor</div>	<div><div></div>Medium</div>
Likelihood (Moderate)	<div><div></div>Informational</div>	<div><div></div>Minor</div>	<div><div></div>Medium</div>	<div><div></div>Major</div>
Likelihood (High)	<div><div></div>Informational</div>	<div><div></div>Medium</div>	<div><div></div>Major</div>	<div><div></div>Major</div>

Unknown Severity

The `unknown` severity level is reserved for misbehaviors we observe in the codebase that cannot be quantified using the above metrics. Examples of such vulnerabilities include potentially desirable system behavior that is undocumented, reliance on external dependencies that are out-of-scope but could result in some form of vulnerability arising, use of external out-of-scope contracts that appears incorrect but cannot be pinpointed, and other such vulnerabilities.

In general, `unknown` severity level vulnerabilities require follow-up information by the project being audited and are either adjusted in severity (if valid), or marked as nullified (if invalid).

Additionally, the `unknown` severity level is sometimes assigned to centralization issues that cannot be assessed in likelihood due to their exploitation being tied to the honesty of the project's team.

Informational Severity

The `informational` severity level is dedicated to findings that do not affect the code functionally and tend to be stylistic or optimizational in nature. Certain edge cases are also set under `informational` vulnerabilities, such as overflow operations that will not manifest in the lifetime of the contract but should be guarded against as a best practice, to give an example.

Minor Severity

The **minor** severity level is meant for vulnerabilities that require functional changes in the code but tend to either have little impact or be unlikely to be recreated in a production environment. These findings can be acknowledged except for findings with a moderate impact but low likelihood which must be alleviated.

Medium Severity

The **medium** severity level is assigned to vulnerabilities that must be alleviated and have an observable impact on the overall project. These findings can only be acknowledged if the project deems them desirable behavior and we disagree with their point-of-view, instead urging them to reconsider their stance while marking the exhibit as acknowledged given that the project has ultimate say as to what vulnerabilities they end up patching in their system.

Major Severity

The **major** severity level is the maximum that can be specified for a finding and indicates a significant flaw in the code that must be alleviated.

Likelihood & Impact Assessment

As the preface chapter specifies, the blockchain space is constantly reinventing itself meaning that new vulnerabilities take place and our understanding of what security means differs year-to-year.

In order to reliably assess the likelihood and impact of a particular vulnerability, we instead apply an abstract measurement of a vulnerability's impact, duration the impact is applied for, and probability that the vulnerability would be exploited in a production environment.

Our proposed definitions are inspired by multiple sources in the security community and are as follows:

- Impact (High): A core invariant of the protocol can be broken for an extended duration.
- Impact (Moderate): A non-core invariant of the protocol can be broken for an extended duration or at scale, or an otherwise major-severity issue is reduced due to hypotheticals or external factors affecting likelihood.
- Impact (Low): A non-core invariant of the protocol can be broken with reduced likelihood or impact.
- Impact (None): A code or documentation flaw whose impact does not achieve low severity, or an issue without theoretical impact; a valuable best-practice
- Likelihood (High): A flaw in the code that can be exploited trivially and is ever-present.
- Likelihood (Moderate): A flaw in the code that requires some external factors to be exploited that are likely to manifest in practice.
- Likelihood (Low): A flaw in the code that requires multiple external factors to be exploited that may manifest in practice but would be unlikely to do so.
- Likelihood (None): A flaw in the code that requires external factors proven to be impossible in a production environment, either due to mathematical constraints, operational constraints, or system-related factors (i.e. EIP-20 tokens not being re-entrant).

Disclaimer

The following disclaimer applies to all versions of the audit report produced (preliminary / public / private) and is in effect for all past, current, and future audit reports that are produced and hosted under Omniscia:

IMPORTANT TERMS & CONDITIONS REGARDING OUR SECURITY AUDITS/REVIEWS/REPORTS AND ALL PUBLIC/PRIVATE CONTENT/DELIVERABLES

Omniscia ("Omniscia") has conducted an independent security review to verify the integrity of and highlight any vulnerabilities, bugs or errors, intentional or unintentional, that may be present in the codebase that were provided for the scope of this Engagement.

Blockchain technology and the cryptographic assets it supports are nascent technologies. This makes them extremely volatile assets. Any assessment report obtained on such volatile and nascent assets may include unpredictable results which may lead to positive or negative outcomes.

In some cases, services provided may be reliant on a variety of third parties. This security review does not constitute endorsement, agreement or acceptance for the Project and technology that was reviewed. Users relying on this security review should not consider this as having any merit for financial advice or technological due diligence in any shape, form or nature.

The veracity and accuracy of the findings presented in this report relate solely to the proficiency, competence, aptitude and discretion of our auditors. Omniscia and its employees make no guarantees, nor assurance that the contracts are free of exploits, bugs, vulnerabilities, deprecation of technologies or any system / economical / mathematical malfunction.

This audit report shall not be printed, saved, disclosed nor transmitted to any persons or parties on any objective, goal or justification without due written assent, acquiescence or approval by Omniscia.

All the information/opinions/suggestions provided in this report does not constitute financial or investment advice, nor should it be used to signal that any person reading this report should invest their funds without sufficient individual due diligence regardless of the findings presented in this report.

Information in this report is provided 'as is'. Omniscia is under no covenant to the completeness, accuracy or solidity of the contracts reviewed. Omniscia's goal is to help reduce the attack vectors/surface and the high level of variance associated with utilizing new and consistently changing technologies.

Omniscia in no way claims any guarantee, warranty or assurance of security or functionality of the technology that was in scope for this security review.

In no event will Omniscia, its partners, employees, agents or any parties related to the design/creation of this security review be ever liable to any parties for, or lack thereof, decisions and/or actions with regards to the information provided in this security review.

Cryptocurrencies and all other technologies directly or indirectly related to cryptocurrencies are not standardized, highly prone to malfunction and extremely speculative by nature. No due diligence and/or safeguards may be insufficient and users should exercise maximum caution when participating and/or investing in this nascent industry.

The preparation of this security review has made all reasonable attempts to provide clear and actionable recommendations to the Project team (the "client") with respect to the rectification, amendment and/or revision of any highlighted issues, vulnerabilities or exploits within the contracts in scope for this engagement.

It is the sole responsibility of the Project team to provide adequate levels of test and perform the necessary checks to ensure that the contracts are functioning as intended, and more specifically to ensure that the functions contained within the contracts in scope have the desired intended effects, functionalities and outcomes, as documented by the Project team.

All services, the security reports, discussions, work product, attack vectors description or any other materials, products or results of this security review engagement is provided "as is" and "as available" and with all faults, uncertainty and defects without warranty or guarantee of any kind.

Omniscia will assume no liability or responsibility for delays, errors, mistakes, or any inaccuracies of content, suggestions, materials or for any loss, delay, damage of any kind which arose as a result of this engagement/security review.

Omniscia will assume no liability or responsibility for any personal injury, property damage, of any kind whatsoever that resulted in this engagement and the customer having access to or use of the products, engineers, services, security report, or any other other materials.

For avoidance of doubt, this report, its content, access, and/or usage thereof, including any associated services or materials, shall not be considered or relied upon as any form of financial, investment, tax, legal, regulatory, or any other type of advice.