Security Review Report
NM-0587 Mellow Protocol

NETHERMIND
SECURITY

(September 03, 2025)

Contents

1 Executive Summary

This document presents the results of the security review conducted by Nethermind Security for Mellow Finance. The Mellow protocol provides a sophisticated and flexible infrastructure for on-chain asset management, enabling curators to design and deploy structured financial products with a high degree of security and control.

The system is built around a central **Vault** contract that orchestrates user deposits, withdrawals, and the delegation of assets to external yield-generating strategies via **Subvaults**. The protocol's architecture is designed to mitigate common DeFi risks such as front-running and price manipulation through time-buffered queues, while providing granular, function-level access control for all strategy-related operations via a novel **Verifier** mechanism.

This audit focused on the end-to-end review of the smart contracts that implement the protocol's core logic, including: **Vault**, **Subvault**, **DepositQueue**, **RedeemQueue**, **Oracle**, **Verifier**, **ShareManager**, **FeeManager**, and **RiskManager**.

The audit comprises 4217 lines of Solidity code. **The audit was performed using** (a) manual analysis of the codebase, and (b) automated analysis tools. **Along this document, we report** 5 points of attention, where two are classified as `High` and three are classified as `Informational` . The issues are summarized in Fig. 1.

This document is organized as follows. Section 2 presents the files in the scope. Section 3 summarizes the issues. Section 4 presents the system overview. Section 5 discusses the risk rating methodology. Section 6 details the issues. Section 7 discusses the documentation provided by the client for this audit. Section 8 presents the test suite evaluation and automated tools used. Section 9 concludes the document.

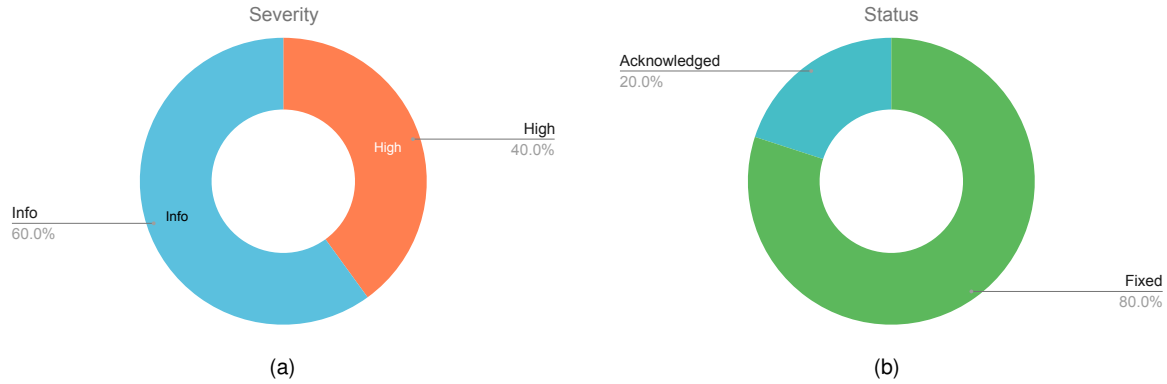

Fig. 1: Distribution of issues: Critical (0), **High** (2), **Medium** (0), **Low** (0), **Undetermined** (0), **Informational** (3), **Best Practices** (0). **Distribution of status: Fixed** (4), **Acknowledged** (1), **Mitigated** (0), **Unresolved** (0)

Summary of the Audit

Audit Type	Security Review
Final Report	September 03, 2025
Repository	mellow-finance/flexible-vaults
Initial Commit	69413d545f788c0ad4ff7fe08085fb55589c5c61
Final Commit	72f689f965e4ac1a4c2bcfb645a8b5416cf740c7
Documentation	Mellow Core Vaults & Flexible Vaults Architecture
Documentation Assessment	High
Test Suite Assessment	High

2 Audited Files

	Contract	LoC	Comments	Ratio	Blank	Total
1	src/permissions/BitmaskVerifier.sol	37	36	97.3%	3	76
2	src/permissions/Consensus.sol	100	14	14.0%	21	135
3	src/permissions/Verifier.sol	137	21	15.3%	27	185
4	src/permissions/MellowACL.sol	45	6	13.3%	13	64
5	src/permissions/protocols/EigenLayerVerifier.sol	122	2	1.6%	8	132
6	src/permissions/protocols/OwnedCustomVerifier.sol	23	2	8.7%	5	30
7	src/permissions/protocols/ERC20Verifier.sol	36	2	5.6%	6	44
8	src/permissions/protocols/SymbioticVerifier.sol	78	2	2.6%	8	88
9	src/strategies/SymbioticStrategy.sol	54	2	3.7%	9	65
10	src/queues/SignatureDepositQueue.sol	18	1	5.6%	5	24
11	src/queues/RedeemQueue.sol	201	12	6.0%	34	247
12	src/queues/SignatureQueue.sol	112	17	15.2%	24	153
13	src/queues/Queue.sol	48	7	14.6%	15	70
14	src/queues/DepositQueue.sol	172	12	7.0%	30	214
15	src/queues/SignatureRedeemQueue.sol	22	1	4.5%	7	30
16	src/managers/TokenizedShareManager.sol	51	7	13.7%	18	76
17	src/managers/RiskManager.sol	206	24	11.7%	34	264
18	src/managers/ShareManager.sol	191	28	14.7%	37	256
19	src/managers/FeeManager.sol	131	20	15.3%	26	177
20	src/managers/BasicShareManager.sol	51	7	13.7%	14	72
21	src/hooks/LidoDepositHook.sol	40	1	2.5%	8	49
22	src/hooks/RedirectingDepositHook.sol	24	1	4.2%	2	27
23	src/hooks/BasicRedeemHook.sol	48	1	2.1%	4	53
24	src/libraries/TransferLibrary.sol	31	23	74.2%	7	61
25	src/libraries/ShareManagerFlagLibrary.sol	26	33	126.9%	8	67
26	src/libraries/SlotLibrary.sol	10	9	90.0%	1	20
27	src/libraries/FenwickTreeLibrary.sol	66	53	80.3%	11	130
28	src/interfaces/permissions/IMellowACL.sol	12	13	108.3%	7	32
29	src/interfaces/permissions/IConsensus.sol	37	45	121.6%	23	105
30	src/interfaces/permissions/IVerifier.sol	74	50	67.6%	32	156
31	src/interfaces/permissions/ICustomVerifier.sol	10	11	110.0%	1	22
32	src/interfaces/queues/IQueue.sol	21	24	114.3%	11	56
33	src/interfaces/queues/IRedeemQueue.sol	44	103	234.1%	14	161
34	src/interfaces/queues/IDepositQueue.sol	31	81	261.3%	17	129
35	src/interfaces/queues/ISignatureQueue.sol	54	79	146.3%	27	160
36	src/interfaces/managers/IRiskManager.sol	61	44	72.1%	40	145
37	src/interfaces/managers/ITokenizedShareManager.sol	6	8	133.3%	1	15
38	src/interfaces/managers/IShareManager.sol	70	62	88.6%	47	179
39	src/interfaces/managers/IFeeManager.sol	42	39	92.9%	24	105
40	src/interfaces/hooks/IHook.sol	4	9	225.0%	1	14
41	src/interfaces/hooks/IRedeemHook.sol	5	7	140.0%	2	14
42	src/interfaces/factories/IFactory.sol	34	33	97.1%	23	90
43	src/interfaces/factories/IFactoryEntity.sol	5	6	120.0%	2	13
44	src/interfaces/oracles/IOracle.sol	65	80	123.1%	38	183
45	src/interfaces/external/symbiotic/ISymbioticRegistry.sol	4	1	25.0%	1	6
46	src/interfaces/external/symbiotic/ISymbioticVault.sol	8	1	12.5%	3	12
47	src/interfaces/external/symbiotic/ISymbioticStakerRewards.sol	7	1	14.3%	4	12
48	src/interfaces/external/eigen-layer/IRewardsCoordinator.sol	90	1	1.1%	43	134
49	src/interfaces/external/eigen-layer/IDelegationManager.sol	86	1	1.2%	33	120
50	src/interfaces/external/eigen-layer/IAllocationManager.sol	118	1	0.8%	45	164
51	src/interfaces/external/eigen-layer/ISignatureUtils.sol	13	1	7.7%	3	17
52	src/interfaces/external/eigen-layer/IStrategy.sol	16	1	6.2%	13	30
53	src/interfaces/external/eigen-layer/IStrategyManager.sol	39	1	2.6%	18	58
54	src/interfaces/external/tokens/IWSTETH.sol	6	1	16.7%	2	9
55	src/interfaces/external/tokens/IWETH.sol	4	1	25.0%	1	6
56	src/interfaces/modules/IVerifierModule.sol	10	7	70.0%	4	21
57	src/interfaces/modules/IBaseModule.sol	8	6	75.0%	2	16
58	src/interfaces/modules/IShareModule.sol	76	52	68.4%	49	177
59	src/interfaces/modules/IACLModule.sol	7	4	57.1%	3	14
60	src/interfaces/modules/IVaultModule.sol	40	62	155.0%	29	131
61	src/interfaces/modules/ISubvaultModule.sol	11	19	172.7%	6	36
62	src/interfaces/modules/ICallModule.sol	9	10	111.1%	2	21
63	src/factories/Factory.sol	92	17	18.5%	22	131

	Contract	LoC	Comments	Ratio	Blank	Total
64	src/oracles/OracleHelper.sol	122	19	15.6%	8	149
65	src/oracles/Oracle.sol	195	23	11.8%	30	248
66	src/vaults/Subvault.sol	16	1	6.2%	3	20
67	src/vaults/VaultConfigurator.sol	65	2	3.1%	7	74
68	src/vaults/Vault.sol	49	1	2.0%	5	55
69	src/modules/VerifierModule.sol	25	4	16.0%	10	39
70	src/modules/SubvaultModule.sol	31	6	19.4%	12	49
71	src/modules/ACLModule.sol	14	2	14.3%	6	22
72	src/modules/CallModule.sol	13	3	23.1%	4	20
73	src/modules/ShareModule.sol	247	39	15.8%	41	327
74	src/modules/VaultModule.sol	124	22	17.7%	26	172
75	src/modules/BaseModule.sol	17	6	35.3%	9	32
	Total	**4217**	**1354**	**32.1%**	**1139**	**6710**

3 Summary of Issues

	Finding	Severity	Update
1	Inconsistent Price Conversion Logic Leads to Incorrect Oracle Prices	High	Fixed
2	Off-chain price calculation can result in an infinite loop	High	Fixed
3	Incorrect Natspec for `CUSTOM_VERIFIER` verification data	Info	Fixed
4	Incorrect natspec for `ClaimableRequestExists` error	Info	Fixed
5	Use of `abi.encodePacked` with dynamic types could lead to storage slot collisions	Info	Acknowledged

4 System Overview

The Mellow protocol introduces a modular and upgradeable vault architecture designed for curators and asset managers to create, deploy, and manage structured on-chain financial products. The system aims to abstract the complexities of DeFi, providing a standardized framework for trustless execution of institutional-grade strategies while maintaining transparency and composability.

The architecture is centered around a main `Vault` contract that serves as the primary user entry point and orchestrates interactions between several specialized modules and peripheral contracts.

Core Components

The system's functionality is segregated into a collection of distinct, interconnected smart contracts:

- **`Vault`**: The central contract that aggregates core logic and serves as the facade for the entire system. It coordinates asset management, share issuance, and interactions with subvaults and queues. It is composed of four main modules: `ACLModule`, `ShareModule`, `VaultModule`, and `BaseModule`.
- **`Subvault`**: An isolated contract responsible for holding assets and executing delegated yield-generating strategies. Curators interact with external protocols through subvaults, with all actions subject to validation by an associated `Verifier`.
- **`DepositQueue` & `RedeemQueue`**: These contracts manage user deposits and withdrawals through a time-buffered queuing system. This mechanism introduces a mandatory delay (`depositInterval` and `redeemInterval`) between a user's request and its execution, which is designed to protect against front-running and price manipulation attacks.
- **`SignatureDepositQueue` & `SignatureRedeemQueue`**: Stateless, alternative queues that allow for instant, gas-efficient deposits and redemptions. They bypass the standard time-delay by processing EIP-712 compliant orders signed by a trusted off-chain consensus group.
- **`Oracle`**: A permissioned price oracle responsible for providing secure and validated price reports for vault assets. It is a critical component for converting assets to shares and vice versa. It includes mechanisms to reject or flag reports based on configurable deviation thresholds.
- **`Verifier`**: A crucial access control component paired with each `Subvault`. It enforces granular, call-level permissions, ensuring that curators can only execute pre-approved interactions with external contracts.
- **`Share Manager`**: An upgradeable contract that handles all logic related to the vault's ERC20 receipt tokens (shares), including issuance, transfers, lockups, and whitelisting.
- **`Fee Manager`**: A centralized module that calculates and directs four types of fees: deposit, redeem, performance, and protocol fees. Fees are typically collected in vault shares.
- **`Risk Manager`**: Enforces configurable deposit limits at both the global `Vault` level and for each individual `Subvault`. It tracks asset balances (denominated in shares) to prevent excessive exposure.
- **`MellowACL`**: A system-wide, role-based access control layer built upon OpenZeppelin's `AccessControl`. It manages permissions for administrative and operational functions.

Key Workflows

User Deposit Workflow

a. A user initiates a deposit by calling `deposit()` on the `DepositQueue`, which registers a request containing the asset amount and a timestamp. Each user is limited to one pending deposit request at a time.

b. The request remains pending until a trusted operator submits a price report to the `Oracle`.

c. An oracle report can only process deposit requests that are older than the configured `depositInterval`. This delay mitigates front-running risks.

d. Upon processing, the system calculates the corresponding number of shares based on the reported price, less any applicable deposit fees. The shares are allocated to the user but not yet minted.

e. The user (or any other address) can then call `claim()` to mint the allocated shares to their account.

User Withdrawal Workflow

a. A user initiates a withdrawal by calling `redeem()` on the `RedeemQueue`, which burns their vault shares and creates a redemption request. To prevent yield-griefing, these requests are non-cancellable.

b. The vault curator is responsible for ensuring sufficient liquidity is available in the main `Vault` to fulfill pending redemptions, potentially by pulling assets from various `Subvault` strategies.

c. A valid `Oracle` report, submitted after the `redeemInterval`, prices the redemption requests.

d. The curator calls `handleBatches()` to process the priced requests, which allocates the underlying assets to the queue for withdrawal.

e. Finally, the user calls `claim()` to receive their underlying assets.

Delegated Strategy Execution

a. A curator, holding the `CALLER_ROLE`, executes an external call (e.g., to an external DeFi protocol) through a `Subvault`.

b. The `Subvault` forwards the call data to its designated `Verifier` contract for authorization.

c. The `Verifier` validates the call against a set of predefined rules based on its verification type:

   - `ONCHAIN_COMPACT`: Checks if a hash of the call's components (`who`, `where`, `selector`) exists in an on-chain allowlist.
   - `MERKLE_COMPACT` / `MERKLE_EXTENDED`: Validates the call against a Merkle root, allowing for a large set of approved calls with minimal on-chain footprint.
   - `CUSTOM_VERIFIER`: Delegates the validation logic to an external contract, enabling complex, state-dependent rules.

d. If the `Verifier` approves the call, the `Subvault` executes it. Otherwise, the transaction reverts.

Security Model

The security of the Mellow Vaults relies on a defense-in-depth approach, combining robust access control, strict price validation, and architectural safeguards.

- **Granular Access Control**: The system employs a two-tiered access control model. High-level administrative powers are managed by `MellowACL`, while low-level curator actions are strictly controlled by the `Verifier` contract, ensuring curators can only perform whitelisted operations.
- **Oracle Security**: The `Oracle` is permissioned, accepting reports only from trusted actors. All submitted prices are validated against configurable absolute and relative deviation thresholds. Reports that are valid but exceed a 'suspicious' threshold are queued for explicit approval by an admin, preventing the system from acting on potentially manipulative price data.
- **Front-Running Mitigation**: The time-delayed nature of the `DepositQueue` and `RedeemQueue` ensures that user actions are priced by future, unknown oracle reports, neutralizing the advantage of front-runners who might otherwise exploit stale prices.
- **Risk Management**: The `Risk Manager` contract enforces system-wide and per-subvault deposit limits, preventing unbounded asset exposure and providing a safeguard against unforeseen protocol risks or economic exploits.

5 Risk Rating Methodology

The risk rating methodology used by Nethermind Security follows the principles established by the OWASP Foundation. The severity of each finding is determined by two factors: **Likelihood** and **Impact**.

Likelihood measures how likely the finding is to be uncovered and exploited by an attacker. This factor will be one of the following values:

a) **High**: The issue is trivial to exploit and has no specific conditions that need to be met;

b) **Medium**: The issue is moderately complex and may have some conditions that need to be met;

c) **Low**: The issue is very complex and requires very specific conditions to be met.

When defining the likelihood of a finding, other factors are also considered. These can include but are not limited to motive, opportunity, exploit accessibility, ease of discovery, and ease of exploit.

Impact is a measure of the damage that may be caused if an attacker exploits the finding. This factor will be one of the following values:

a) **High**: The issue can cause significant damage, such as loss of funds or the protocol entering an unrecoverable state;

b) **Medium**: The issue can cause moderate damage, such as impacts that only affect a small group of users or only a particular part of the protocol;

c) **Low**: The issue can cause little to no damage, such as bugs that are easily recoverable or cause unexpected interactions that cause minor inconveniences.

When defining the impact of a finding, other factors are also considered. These can include but are not limited to Data/state integrity, loss of availability, financial loss, and reputation damage. After defining the likelihood and impact of an issue, the severity can be determined according to the table below.

		Severity Risk		
Impact	**High**	Medium	High	Critical
	Medium	Low	Medium	High
	Low	Info/Best Practices	Low	Medium
	Undetermined	Undetermined	Undetermined	Undetermined
		Low	**Medium**	**High**
		Likelihood		

To address issues that do not fit a High/Medium/Low severity, Nethermind Security also uses three more finding severities: **Informational**, **Best Practices**, and **Undetermined**.

a) **Informational** findings do not pose any risk to the application, but they carry some information that the audit team intends to pass to the client formally;

b) **Best Practice** findings are used when some piece of code does not conform with smart contract development best practices;

c) **Undetermined** findings are used when we cannot predict the impact or likelihood of the issue.

6 Issues

6.1 [High] Inconsistent Price Conversion Logic Leads to Incorrect Oracle Prices

File(s): src/oracles/OracleHelper.sol

Description: The `OracleHelper` contract is responsible for calculating the prices of assets held within a vault, which are then reported to an oracle. The price of each asset is defined relative to a "base asset" through the `priceD18` parameter, as explained in the comments of the `AssetPrice` struct. According to these comments, to convert an amount from a specific asset to the base asset, one should multiply by `priceD18`.

The `getPricesD18(...)` function performs two key calculations using `priceD18` that are logically inconsistent with each other.

First, when calculating the `totalRedeemDemand` (the total value of withdrawal demands in terms of the base asset), the code divides the `demand_` by `assetPrice.priceD18`. This contradicts the logic described in the comments.

```
// ...
// @audit This comment explains that priceD18 is a multiplier.
// @audit - If priceD18 = 2e18, it means that 1 asset = 2 base assets
/**
 * Price of the asset expressed via the base asset.
// ...
 */

// ...
$.unprocessedShares += shares_;
if (assetPrice.priceD18 == 0) {
    $.totalRedeemDemand += demand_;
} else {
    // @audit-issue The demand is divided by the price, not multiplied.
    // @audit-issue This should be Math.mulDiv(demand_, assetPrice.priceD18, 1 ether).
    $.totalRedeemDemand += Math.mulDiv(demand_, 1 ether, assetPrice.priceD18);
}
// ...
```

Later in the same function, when calculating the final prices of the other assets, the logic correctly multiplies the base asset's price by `assetPrices[i].priceD18`. This second calculation is consistent with the comments.

```
// ...
for (uint256 i = 0; i < assetPrices.length; i++) {
    if (i != $.baseAssetIndex) {
        // @audit This calculation correctly uses priceD18 as a multiplier.
        pricesD18[i] = Math.mulDiv(
            pricesD18[$.baseAssetIndex], assetPrices[i].priceD18, 1 ether
        );
    }
// ...
```

The incorrect calculation of `totalRedeemDemand` will lead to a miscalculation of the base asset's price. Since all other asset prices are derived from the base asset price, the entire set of prices returned by the function will be incorrect. This can severely compromise the protocol's economic security, leading to incorrect vault valuations, unfair distribution of funds upon withdrawal, and potential financial exploits.

Recommendation(s): Review the intended formula for converting asset amounts to their base asset equivalents. Assuming the comments and the final price calculation logic are correct, the calculation for `totalRedeemDemand` should be changed from division to multiplication.

Status: Fixed

Update from the client: Fixed in 72f689f965e4ac1a4c2bcfb645a8b5416cf740c7

6.2 [High] Off-chain price calculation can result in an infinite loop

File(s): src/oracles/OracleHelper.sol, src/managers/FeeManager.sol

Description: The `OracleHelper` contract includes the `getPricesD18(...)` function, which is designed for off-chain execution by keepers to calculate a vault's share price. This calculation is iterative, using a `while(true)` loop to find a stable price that correctly accounts for performance and protocol fees. The loop is expected to terminate once the calculated price converges.

The performance fee calculation, handled by the `FeeManager` contract, is conditional. It is applied only when the share price surpasses its previous high water mark, which corresponds to when `priceD18` is less than `minPriceD18`. This conditional logic can create a scenario where the calculated price oscillates between two values, preventing convergence and causing the loop to run indefinitely.

This issue arises when an iteration calculates a price (`P1`) low enough to trigger the performance fee. In the next iteration, the inclusion of this fee results in a higher calculated price (`P2`). If `P2` is now above the high-water mark, the performance fee is no longer applied in the subsequent iteration, causing the calculated price to revert to `P1`. This cycle of price oscillation between `P1` and `P2` means the loop's exit condition is never met.

The core of the issue lies in the `while` loop in the `OracleHelper` contract.

```
// src/oracles/OracleHelper.sol

// ...
        uint256 baseAssetPriceD18 = vault.oracle().getReport($.baseAsset).priceD18;
// @audit The loop lacks a circuit breaker and can run indefinitely under certain conditions.
        while (true) {
            uint256 feeShares = totalShares > recipientShares
                ? feeManager.calculateFee(
                    address(vault), $.baseAsset, baseAssetPriceD18, totalShares - recipientShares
                )
                : 0;
            uint256 newBaseAssetPriceD18 = Math.mulDiv(
                totalShares + $.unprocessedShares + feeShares,
                1 ether,
                totalAssets - $.totalRedeemDemand
            );
// @audit-issue This equality check may never be true if the price oscillates.
            if (newBaseAssetPriceD18 == baseAssetPriceD18) {
                break;
            }
            baseAssetPriceD18 = newBaseAssetPriceD18;
        }
// ...
```

The conditional logic in the `FeeManager` contract.

```
// src/managers/FeeManager.sol
function calculateFee(address vault, address asset, uint256 priceD18, uint256 totalShares)
    public
    view
    returns (uint256 shares)
{
// ...
        if (asset == $.baseAsset[vault]) {
            uint256 minPriceD18_ = $.minPriceD18[vault];
            // @audit Performance fees are only added if the price is below the high-water mark.
            if (priceD18 < minPriceD18_ && minPriceD18_ != 0) {
                shares = Math.mulDiv(
                    minPriceD18_ - priceD18,
                    $.performanceFeeD6 * totalShares,
                    priceD18 * 1e6
                );
            }
// ...
        }
}
```

An infinite loop in the off-chain price calculation script will cause it to fail, preventing keepers from submitting updated price reports. This leads to stale prices on-chain, which can halt critical vault operations or be exploited by attackers, resulting in a Denial of Service for the price update mechanism.

Recommendation(s): Consider re-architecting the iterative price calculation to guarantee convergence.

Status: Fixed

Update from the client: Fixed in 72f689f965e4ac1a4c2bcfb645a8b5416cf740c7

6.3 [Info] Incorrect Natspec for `CUSTOM_VERIFIER` verification data

File(s): src/permissions/Verifier.sol

Description: The `getVerificationResult(...)` function in the `Verifier` contract handles the authorization of delegated function calls based on different verification types. One such type is `CUSTOM_VERIFIER`, which allows for verification logic to be delegated to an external contract.

The natspec documentation for the `VerificationPayload` struct provides instructions on how to encode the `verificationData` for each verification type. For `CUSTOM_VERIFIER`, it incorrectly suggests using `abi.encodePacked(address customVerifier, customVerifierSpecificData)`.

```
    /// @notice Struct containing all inputs required to verify a delegated function call.
    struct VerificationPayload {
        /// @dev The method used to verify the delegated call.
        VerificationType verificationType;
        /// @dev Encoded payload to be verified, varies by verification type:
        /// - MERKLE_COMPACT: abi.encode(who, where, selector)
        /// - MERKLE_EXTENDED: abi.encode(who, where, value, callData)
        // @audit-issue Natspec suggests abi.encodePacked, which is incompatible with the implementation.
        /// - CUSTOM_VERIFIER: abi.encodePacked(address customVerifier, customVerifierSpecificData)
        bytes verificationData;
        /// @dev Merkle proof used to validate the `verificationType` and `verificationData` for MERKLE_COMPACT,
        /// MERKLE_EXTENDED, and CUSTOM_VERIFIER types.
        bytes32[] proof;
    }
```

The issue arises in the `getVerificationResult(...)` function, where the `verifier` address is extracted from `verificationData` using an assembly block with the `calldataload` instruction. This instruction reads a full 32-byte word.

```
    function getVerificationResult(...) public view returns (bool) {
// ...
        } else if (payload.verificationType == VerificationType.CUSTOM_VERIFIER) {
            address verifier;
            assembly {
                // @audit This instruction reads a full 32-byte word from calldata.
                verifier := calldataload(verificationData.offset)
            }
            return ICustomVerifier(verifier).verifyCall(who, where, value, data, verificationData[0x20:]);
// ...
    }
```

If a developer follows the natspec and uses `abi.encodePacked`, the 20-byte `verifier` address will be tightly packed with `customVerifierSpecificData`. The `calldataload` instruction would then read the address plus the first 12 bytes of the custom data, resulting in a corrupted address and causing the verification to always fail. The correct encoding should pad the address to 32 bytes, for example, by using `abi.encode(...)`.

This discrepancy between the documentation and the implementation can mislead developers, causing them to create invalid payloads and breaking the `CUSTOM_VERIFIER` functionality for anyone who relies on the natspec.

Recommendation(s): Consider updating the natspec comment for `verificationData` in the `VerificationPayload` struct. Instead of `abi.encodePacked(...)`, it should specify `abi.encode(...)` or another encoding method that ensures the `customVerifier` address is padded to a full 32-byte word to align with the implementation.

Status: Fixed

Update from the client: Fixed in 72f689f965e4ac1a4c2bcfb645a8b5416cf740c7

6.4 [Info] Incorrect natspec for `ClaimableRequestExists` error

File(s): `src/queues/DepositQueue.sol`, `src/interfaces/queues/IDepositQueue.sol`

Description: The `cancelDepositRequest(...)` function in the `DepositQueue` contract allows a user to cancel their pending deposit request. The function checks if the request has become claimable by comparing the request's timestamp with the latest price checkpoint timestamp. If the request is claimable, the function reverts with the `ClaimableRequestExists` error.

```
// src/queues/DepositQueue.sol

function cancelDepositRequest() external nonReentrant {
    // ...
    (bool exists, uint32 timestamp,) = $.prices.latestCheckpoint();
    // @audit The function reverts if a user tries to cancel a request that is already claimable.
    if (exists && timestamp >= request._key) {
        revert ClaimableRequestExists();
    }
    // ...
}
```

However, the natspec documentation for the `ClaimableRequestExists` error in the `IDepositQueue` interface is incorrect. It states that the error is thrown when a user tries to deposit again, which is not the case.

```
// src/interfaces/queues/IDepositQueue.sol

// @audit-issue The natspec incorrectly states the error is for depositing again.
/// @notice Thrown when a user tries to deposit again while a claimable request exists.
error ClaimableRequestExists();
```

This discrepancy can mislead developers, auditors, and integrators about the actual behavior of the `cancelDepositRequest(...)` function, potentially causing confusion when interacting with the contract.

Recommendation(s): Consider updating the natspec for the `ClaimableRequestExists` error to accurately reflect its usage. The comment should clarify that the error is thrown when a user attempts to cancel a deposit request that has already become claimable.

Status: Fixed

Update from the client: Fixed in 72f689f965e4ac1a4c2bcfb645a8b5416cf740c7

6.5 [Info] Use of `abi.encodePacked` with dynamic types could lead to storage slot collisions

File(s): src/libraries/SlotLibrary.sol

Description: The `SlotLibrary` contract provides a helper function, `getSlot(...)`, to compute deterministic storage slots for various modules based on a `contractName`, an instance `name`, and a `version`. This mechanism is used by contracts such as `VerifierModule` and `Verifier` to derive their unique storage locations.

The implementation of `getSlot(...)` uses `abi.encodePacked(...)` to concatenate multiple dynamic `string` arguments. This function is known to be unsafe when used with multiple dynamic types, as it can lead to hash collisions. For example, the following two different sets of inputs would produce the same output:

- `abi.encodePacked("VerifierModule", "example", 1);`
- `abi.encodePacked("Verifier", "Moduleexample", 1);`

This is because `abi.encodePacked` directly concatenates the byte representations of the strings without any padding or delimiters.

```
function getSlot(string memory contractName, string memory name, uint256 version)
    internal
    pure
    returns (bytes32)
{
    // @audit-issue Use of abi.encodePacked with multiple dynamic types can lead to collisions.
    return keccak256(
        abi.encode(
            uint256(
                keccak256(
                    abi.encodePacked(
                        "mellow.flexible-vaults.storage.",
                        contractName,
                        name,
                        version
                    )
                )
            ) - 1
        )
    ) & ~bytes32(uint256(0xff));
}
```

This vulnerability could allow two different modules, for instance, a `VerifierModule` with `name` set to `"example"` and a `Verifier` with `name` set to `"Moduleexample"`, to be assigned the same storage slot if they share the same version. This would cause the storage of one module to overwrite the other, leading to state corruption and unpredictable behavior.

Recommendation(s): Consider revisiting the slot generation logic in light of the potential for hash collisions with `abi.encodePacked`. While the risk may be low, it is important to be aware of this behavior. Adding explanatory comments to the function could also serve to inform future developers of this subtlety.

Status: Acknowledged

Update from the client: Acknowledged, not applicable. This issue is mitigated by the current assumptions: `contractName` always corresponds to the actual contract name, and `name` is static, so storage collisions are impossible.

7 Documentation Evaluation

Software documentation refers to the written or visual information that describes the functionality, architecture, design, and implementation of software. It provides a comprehensive overview of the software system and helps users, developers, and stakeholders understand how the software works, how to use it, and how to maintain it. Software documentation can take different forms, such as user manuals, system manuals, technical specifications, requirements documents, design documents, and code comments. Software documentation is critical in software development, enabling effective communication between developers, testers, users, and other stakeholders. It helps to ensure that everyone involved in the development process has a shared understanding of the software system and its functionality. Moreover, software documentation can improve software maintenance by providing a clear and complete understanding of the software system, making it easier for developers to maintain, modify, and update the software over time. Smart contracts can use various types of software documentation. Some of the most common types include:

- Technical whitepaper: A technical whitepaper is a comprehensive document describing the smart contract's design and technical details. It includes information about the purpose of the contract, its architecture, its components, and how they interact with each other;
- User manual: A user manual is a document that provides information about how to use the smart contract. It includes step-by-step instructions on how to perform various tasks and explains the different features and functionalities of the contract;
- Code documentation: Code documentation is a document that provides details about the code of the smart contract. It includes information about the functions, variables, and classes used in the code, as well as explanations of how they work;
- API documentation: API documentation is a document that provides information about the API (Application Programming Interface) of the smart contract. It includes details about the methods, parameters, and responses that can be used to interact with the contract;
- Testing documentation: Testing documentation is a document that provides information about how the smart contract was tested. It includes details about the test cases that were used, the results of the tests, and any issues that were identified during testing;
- Audit documentation: Audit documentation includes reports, notes, and other materials related to the security audit of the smart contract. This type of documentation is critical in ensuring that the smart contract is secure and free from vulnerabilities.

These types of documentation are essential for smart contract development and maintenance. They help ensure that the contract is properly designed, implemented, and tested, and they provide a reference for developers who need to modify or maintain the contract in the future.

Remarks about Mellow Protocol's documentation

The `Mellow Protocol` team provided an overview of the main system components during the kick-off call with a detailed explanation of the intended functionalities. They also supplied Notion documentation detailing the contracts' inner workings and flow: Mellow Core Vaults & Flexible Vaults Architecture. Moreover, the team addressed all questions and concerns from the Nethermind Security team, offering valuable insights into the project's technical aspects.

8 Test Suite Evaluation

8.1 Compilation Output

```
> yarn compile
yarn run v1.22.22
$ forge build --use 0.8.25 --cache-path cache
[] Compiling...
[] Compiling 201 files with Solc 0.8.25
[] Solc 0.8.25 finished in 44.06s
Compiler run successful!
```

8.2 Tests Output

```
> yarn test

yarn run v1.22.22
$ forge test --fork-url $(grep ETH_RPC .env | cut -d '=' -f2,3,4,5) --gas-limit 10000000000000000 --fork-block-number
↪  22730425 -vvv
No files changed, compilation skipped

Ran 1 test for test/unit/libraries/TransferLibrary.t.sol:TransferWrapper
[PASS] test() (gas: 139)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 12.28ms (2.75ms CPU time)

Ran 1 test for test/unit/factories/Factory.t.sol:Mock
[PASS] test() (gas: 185)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 12.29ms (2.70ms CPU time)

Ran 1 test for test/mocks/MockShareManager.sol:MockShareManager
[PASS] test() (gas: 185)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 12.28ms (2.57ms CPU time)

Ran 1 test for test/unit/modules/ACLModule.t.sol:ACLModuleTest
[PASS] testRole() (gas: 1642214)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 13.06ms (3.34ms CPU time)

Ran 3 tests for test/unit/modules/BaseModule.t.sol:BaseModuleTest
[PASS] testCreate() (gas: 873177)
[PASS] testGetStorageAt() (gas: 876462)
[PASS] testOnERC721Received() (gas: 878002)
Suite result: ok. 3 passed; 0 failed; 0 skipped; finished in 13.52ms (3.80ms CPU time)

Ran 1 test for test/mocks/MockSubvault.sol:MockSubvault
[PASS] test() (gas: 119)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 709.72µs (70.39µs CPU time)

Ran 1 test for test/unit/modules/BaseModule.t.sol:MockBaseModule
[PASS] test() (gas: 161)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 385.67µs (75.78µs CPU time)

Ran 1 test for test/mocks/MockVault.sol:MockVault
[PASS] test() (gas: 227)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 649.27µs (68.40µs CPU time)

Ran 2 tests for test/unit/modules/CallModule.t.sol:CallModuleTest
[PASS] testCreate() (gas: 3511135)
[PASS] testVerificationCall() (gas: 3669049)
Suite result: ok. 2 passed; 0 failed; 0 skipped; finished in 15.65ms (3.07ms CPU time)

Ran 6 tests for test/unit/hooks/BasicRedeemHook.t.sol:BasicRedeemHookTest
[PASS] testBeforeRedeem() (gas: 300347)
[PASS] testBeforeRedeem_WithNativeToken() (gas: 122062)
[PASS] testBeforeRedeem_WithNotAllowedAsset() (gas: 287404)
[PASS] testGetLiquidAssets() (gas: 281353)
[PASS] testGetLiquidAssets_WithNativeToken() (gas: 101846)
[PASS] testGetLiquidAssets_WithNotAllowedAsset() (gas: 432169)
Suite result: ok. 6 passed; 0 failed; 0 skipped; finished in 15.67ms (5.95ms CPU time)
```

```
Ran 1 test for test/unit/queues/Nethermind.t.sol:NethermindRedeemQueueTest
[PASS] test() (gas: 185)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 2.44ms (69.35µs CPU time)

Ran 26 tests for test/unit/vaults/Vault.t.sol:VaultTest
[PASS] testConstructorSetsDepositQueueFactory() (gas: 15961)
[PASS] testConstructorSetsRedeemQueueFactory() (gas: 15918)
[PASS] testConstructorSetsSubvaultFactory() (gas: 15926)
[PASS] testConstructorSetsUniqueACLModuleSlot() (gas: 4562420)
[PASS] testConstructorSetsUniqueShareModuleSlot() (gas: 4559050)
[PASS] testConstructorSetsUniqueVaultModuleSlot() (gas: 4559115)
[PASS] testConstructorSetsVerifierFactory() (gas: 15883)
[PASS] testInitializeEmitsInitializedEvent() (gas: 4410746)
[PASS] testInitializeGrantsRegularRoles() (gas: 4698142)
[PASS] testInitializeHandlesEmptyRoleHolders() (gas: 4403993)
[PASS] testInitializeHandlesSameAccountMultipleRoles() (gas: 4696698)
[PASS] testInitializeRevertsIfCalledTwice() (gas: 4415536)
[PASS] testInitializeRevertsOnInsufficientParameters() (gas: 3599416)
[PASS] testInitializeRevertsOnZeroAdminAddress() (gas: 3599430)
[PASS] testInitializeRevertsOnZeroFeeManagerAddress() (gas: 3739585)
[PASS] testInitializeRevertsOnZeroOracleAddress() (gas: 3739571)
[PASS] testInitializeRevertsOnZeroRiskManagerAddress() (gas: 3872782)
[PASS] testInitializeRevertsOnZeroShareManagerAddress() (gas: 3739599)
[PASS] testInitializeSetsAdmin() (gas: 19440)
[PASS] testInitializeSetsDefaultDepositHook() (gas: 18044)
[PASS] testInitializeSetsDefaultRedeemHook() (gas: 18110)
[PASS] testInitializeSetsFeeManager() (gas: 18047)
[PASS] testInitializeSetsOracle() (gas: 18134)
[PASS] testInitializeSetsQueueLimit() (gas: 15866)
[PASS] testInitializeSetsRiskManager() (gas: 18082)
[PASS] testInitializeSetsShareManager() (gas: 18111)
Suite result: ok. 26 passed; 0 failed; 0 skipped; finished in 16.53ms (12.94ms CPU time)

Ran 3 tests for test/unit/managers/BasicShareManager.t.sol:BasicShareManagerTest
[PASS] test() (gas: 185)
[PASS] testCreate() (gas: 35899319)
[PASS] testMintAndBurn() (gas: 37777741)
Suite result: ok. 3 passed; 0 failed; 0 skipped; finished in 20.23ms (17.63ms CPU time)

Ran 2 tests for test/unit/permissions/BitmaskVerifier.t.sol:BitmaskVerifierTest
[PASS] testInvalid() (gas: 312083)
[PASS] testValid() (gas: 343563)
Logs:
  0x0b5abfb44d8f4c0d87692fc7abded4d62338e850f213553d30c3d762c9dea21a
  0x00000000000000000000000fffffffffffffffffffffffffffffffffffffffff0000000000000000000000000fffffffffffffffffffffff
  ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff0000000000000000000000000000000
  0000000000000000000000000000000000000000
  0x350c530b0000000000000000000000000000000000000000000000000000000000000000002a
  36 132

Suite result: ok. 2 passed; 0 failed; 0 skipped; finished in 2.80ms (2.17ms CPU time)

Ran 1 test for test/integration/SymbioticFlow.t.sol:SymbioticIntegrationTest
[PASS] testSymbioticFlow() (gas: 13228309)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 40.76ms (17.95ms CPU time)

Ran 3 tests for test/unit/vaults/VaultConfigurator.t.sol:VaultConfiguratorTest
[PASS] testConstructorSetsFactoryAddresses() (gas: 24204)
[PASS] testCreateSetsVaultRelationships() (gas: 3928823)
[PASS] testCreateSuccessfullyCreatesAllComponents() (gas: 3943457)
Suite result: ok. 3 passed; 0 failed; 0 skipped; finished in 10.61ms (5.03ms CPU time)

Ran 1 test for test/unit/libraries/FenwickTreeLibrary.t.sol:FenwickWrapper
[PASS] test() (gas: 185)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 305.37µs (96.58µs CPU time)

Ran 1 test for test/integration/SymbioticFlowWithSlashing.t.sol:SymbioticWithSlashingIntegrationTest
[PASS] testSymbioticFlowWithSlashing() (gas: 14747482)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 37.46ms (24.24ms CPU time)
```

```
Ran 2 tests for test/unit/queues/Queue.t.sol:QueueTest
[PASS] testCreate() (gas: 1333990)
[PASS] testHandleReport() (gas: 1219580)
Suite result: ok. 2 passed; 0 failed; 0 skipped; finished in 2.09ms (1.31ms CPU time)

Ran 10 tests for test/unit/permissions/Verifier.t.sol:VerifierTest
[PASS] testAllowCalls() (gas: 2816811)
[PASS] testCreate() (gas: 2651633)
[PASS] testDisallowCalls() (gas: 2713522)
[PASS] testInitialize() (gas: 2071533)
[PASS] testSetMerkleRoot() (gas: 2656650)
[PASS] testVerificationCall_Custom_Verifier_Fail() (gas: 2810696)
[PASS] testVerificationCall_Custom_Verifier_Success() (gas: 2810200)
[PASS] testVerificationCall_Merkle_Compat() (gas: 2702924)
[PASS] testVerificationCall_Merkle_Extended() (gas: 2686812)
[PASS] testVerificationCall_Onchain_Compat() (gas: 2813669)
Suite result: ok. 10 passed; 0 failed; 0 skipped; finished in 19.83ms (17.44ms CPU time)

Ran 1 test for test/unit/modules/VerifierModule.t.sol:VerifierModuleTest
[PASS] testVerifierModule() (gas: 939044)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 2.36ms (1.74ms CPU time)

Ran 5 tests for test/unit/modules/VaultModule.t.sol:VaultModuleTest
[PASS] test() (gas: 207)
[PASS] testCreateSubvault() (gas: 40602820)
[PASS] testDisconnectSubvault() (gas: 37118786)
[PASS] testPushAndPullAssets() (gas: 37454878)
[PASS] testReconnectSubvault() (gas: 79799027)
Suite result: ok. 5 passed; 0 failed; 0 skipped; finished in 64.08ms (51.64ms CPU time)

Ran 1 test for test/unit/permissions/Verifier.t.sol:MockCustomVerifier
[PASS] test() (gas: 119)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 555.20µs (119.27µs CPU time)

Ran 1 test for test/integration/ProtocolFees.t.sol:IntegrationTest
[PASS] testProtocolFees() (gas: 8929970)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 23.53ms (11.25ms CPU time)

Ran 7 tests for test/unit/managers/FeeManager.t.sol:FeeManagerTest
[PASS] test() (gas: 207)
[PASS] testCreate() (gas: 36385585)
[PASS] testFeeCalculation() (gas: 36467025)
[PASS] testFeeCalculation_NotBaseAsset() (gas: 36920898)
[PASS] testFeeCalculation_PerformanceFeeArithmetic() (gas: 36449515)
[PASS] testOnlyOwner() (gas: 36371484)
[PASS] testSetFeesAndAsset() (gas: 36421462)
Suite result: ok. 7 passed; 0 failed; 0 skipped; finished in 60.50ms (57.70ms CPU time)

Ran 2 tests for test/unit/hooks/LidoDepositHook.t.sol:LidoDepositHookTest
[PASS] testAfterDeposit() (gas: 1095420)
[PASS] testAfterDepositNextHook() (gas: 956509)
Suite result: ok. 2 passed; 0 failed; 0 skipped; finished in 10.90ms (8.71ms CPU time)

Ran 4 tests for test/unit/permissions/MellowACL.t.sol:MellowACLTest
[PASS] testGrantRole() (gas: 1789437)
[PASS] testInitialize() (gas: 1483588)
[PASS] testOnlySelfOrRole() (gas: 1621347)
[PASS] testRevokeRole() (gas: 1557023)
Suite result: ok. 4 passed; 0 failed; 0 skipped; finished in 3.93ms (2.78ms CPU time)

Ran 4 tests for test/unit/managers/TokenizedShareManager.t.sol:TokenizedShareManagerTest
[PASS] test() (gas: 185)
[PASS] testConstructorSetsUniqueStorageSlots() (gas: 36509015)
[PASS] testCreate() (gas: 36550550)
[PASS] testShouldClaimSharesOnTransfer() (gas: 36569112)
Suite result: ok. 4 passed; 0 failed; 0 skipped; finished in 29.83ms (27.06ms CPU time)

Ran 1 test for test/integration/SymbioticFlowWithSlashingWithRedeemFee.t.sol:SymbioticWithSlashingIntegrationTest
[PASS] testSymbioticFlowWithSlashingWithRedemptionFee() (gas: 16294042)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 53.94ms (41.25ms CPU time)
```

```
Ran 4 tests for test/unit/libraries/TransferLibrary.t.sol:TransferLibraryTest
[PASS] testBalanceOf(uint128,uint128) (runs: 258, : 41079, ~: 41695)
[PASS] testReceiveETHWrongAmount() (gas: 17961)
[PASS] testSendAndReceiveERC20() (gas: 61969)
[PASS] testSendAndReceiveETH() (gas: 27696)
Suite result: ok. 4 passed; 0 failed; 0 skipped; finished in 66.55ms (65.45ms CPU time)

Ran 29 tests for test/unit/permissions/protocols/SymbioticVerifier.t.sol:SymbioticVerifierTest
[PASS] testInitializeCorrectlyGrantsAdminRole() (gas: 2063753)
[PASS] testInitializeCorrectlyGrantsRoles() (gas: 2494828)
[PASS] testInitializeRevertsIfCalledTwice() (gas: 19838)
[PASS] testInitializeRevertsOnZeroAdmin() (gas: 1278127)
[PASS] testInitializeRevertsOnZeroHolder() (gas: 1415941)
[PASS] testInitializeRevertsOnZeroRole() (gas: 1417618)
[PASS] testInitializeWithArrayLengthMismatchMoreHolders() (gas: 1558675)
[PASS] testVerifyCallClaim(uint256) (runs: 258, : 29719, ~: 29719)
[PASS] testVerifyCallClaimRevertsOnInvalidRecipient() (gas: 28437)
[PASS] testVerifyCallClaimRevertsOnMalformedCallData() (gas: 29857)
[PASS] testVerifyCallClaimRewards() (gas: 34843)
[PASS] testVerifyCallClaimRewardsRevertsOnInvalidRecipient() (gas: 33374)
[PASS] testVerifyCallClaimRewardsRevertsOnMalformedCallData() (gas: 35000)
[PASS] testVerifyCallClaimRewardsRevertsOnZeroToken() (gas: 32121)
[PASS] testVerifyCallDeposit(uint256) (runs: 258, : 29980, ~: 29980)
[PASS] testVerifyCallDepositRevertsOnInvalidOnBehalfOf() (gas: 28278)
[PASS] testVerifyCallDepositRevertsOnMalformedCallData() (gas: 29766)
[PASS] testVerifyCallDepositRevertsOnZeroAmount() (gas: 29110)
[PASS] testVerifyCallIgnoresVerificationData(uint256) (runs: 258, : 34982, ~: 34982)
[PASS] testVerifyCallRevertsOnInsufficientCallDataLength() (gas: 19113)
[PASS] testVerifyCallRevertsOnNonZeroValue() (gas: 21471)
[PASS] testVerifyCallRevertsOnUnauthorizedCaller() (gas: 23045)
[PASS] testVerifyCallRevertsOnUnknownContract() (gas: 27941)
[PASS] testVerifyCallSymbioticFarmRevertsOnUnknownSelector() (gas: 31038)
[PASS] testVerifyCallSymbioticVaultRevertsOnUnknownSelector() (gas: 26395)
[PASS] testVerifyCallWithdraw(uint256) (runs: 258, : 29965, ~: 29965)
[PASS] testVerifyCallWithdrawRevertsOnInvalidClaimer() (gas: 28349)
[PASS] testVerifyCallWithdrawRevertsOnMalformedCallData() (gas: 29837)
[PASS] testVerifyCallWithdrawRevertsOnZeroAmount() (gas: 29182)
Suite result: ok. 29 passed; 0 failed; 0 skipped; finished in 183.68ms (170.15ms CPU time)

Ran 2 tests for test/unit/hooks/RedirectingDepositHook.t.sol:RedirectingDepositHookTest
[PASS] testPush() (gas: 355521)
[PASS] testPushSkip() (gas: 321729)
Suite result: ok. 2 passed; 0 failed; 0 skipped; finished in 1.64ms (779.94µs CPU time)

Ran 16 tests for test/unit/libraries/ShareManagerFlagLibrary.t.sol:ShareManagerFlagLibraryTest
[PASS] testCreateMaskAllBooleanFlags() (gas: 5347)
[PASS] testCreateMaskAllZero() (gas: 5367)
[PASS] testCreateMaskWithLockups() (gas: 5341)
[PASS] testGetGlobalLockupCombinedWithFlags() (gas: 3104)
[PASS] testGetGlobalLockupFuzzed(uint256) (runs: 258, : 3216, ~: 3216)
[PASS] testGetGlobalLockupIgnoreHigherBits() (gas: 3210)
[PASS] testGetGlobalLockupMaxValue() (gas: 3145)
[PASS] testGetGlobalLockupOnlyFlags() (gas: 3106)
[PASS] testGetGlobalLockupSimpleValue() (gas: 3162)
[PASS] testGetGlobalLockupZero() (gas: 3155)
[PASS] testHasBurnPause(uint256) (runs: 258, : 5785, ~: 5785)
[PASS] testHasMintPause(uint256) (runs: 258, : 5827, ~: 5827)
[PASS] testHasTransferPause(uint256) (runs: 258, : 5756, ~: 5756)
[PASS] testHasTransferWhitelist(uint256) (runs: 258, : 5833, ~: 5833)
[PASS] testHasWhitelist(uint256) (runs: 258, : 5811, ~: 5811)
[PASS] testMaskApplicationFuzzed(bool,bool,bool,bool,bool,uint32) (runs: 258, : 8064, ~: 8069)
Suite result: ok. 16 passed; 0 failed; 0 skipped; finished in 154.00ms (153.54ms CPU time)

Ran 4 tests for test/unit/queues/SignatureQueue.t.sol:SignatureQueueTest
[PASS] test() (gas: 207)
[PASS] testCreate() (gas: 39178706)
[PASS] testValidateOrder() (gas: 41000659)
[PASS] testValidateOrder_WithPriceRounding() (gas: 41008113)
Suite result: ok. 4 passed; 0 failed; 0 skipped; finished in 117.37ms (113.66ms CPU time)
```

```
Ran 13 tests for test/unit/modules/ShareModule.t.sol:ShareModuleTest
[PASS] test() (gas: 185)
[PASS] testCallHookForbidden() (gas: 39041497)
[PASS] testCreate() (gas: 36405443)
[PASS] testCreateDepositQueue() (gas: 37478959)
[PASS] testCreateRedeemQueue() (gas: 37436617)
[PASS] testGetHook() (gas: 38907508)
[PASS] testGetLiquidAssets() (gas: 38265448)
[PASS] testGetLiquidAssets_WithERC20VaultBalanceWhenNoHook() (gas: 37392337)
[PASS] testGetLiquidAssets_WithNativeVaultBalanceWhenNoHook() (gas: 37334644)
[PASS] testHandleReport() (gas: 37570614)
[PASS] testPauseQueue() (gas: 38366422)
[PASS] testRemoveQueue() (gas: 38285775)
[PASS] testSetCustomHook() (gas: 38941735)
Suite result: ok. 13 passed; 0 failed; 0 skipped; finished in 215.20ms (212.66ms CPU time)

Ran 12 tests for test/unit/queues/Queue2.t.sol:QueueTest2
[PASS] testAssetReturnExpectedValue(uint160) (runs: 258, : 1212856, ~: 1212856)
[PASS] testConstructorSetsUniqueStorageSlot() (gas: 13228)
[PASS] testHandleReportCallsInternalHook(uint224,uint32) (runs: 258, : 23369, ~: 23369)
[PASS] testHandleReportRevertsForNonVaultCaller() (gas: 15697)
[PASS] testHandleReportRevertsOnFutureTimestamp() (gas: 27311)
[PASS] testHandleReportRevertsOnZeroPrice() (gas: 18315)
[PASS] testHandleReportSuccess(uint224,uint32) (runs: 258, : 23413, ~: 23413)
[PASS] testInitializeRevertsIfCalledTwice() (gas: 20408)
[PASS] testInitializeRevertsOnZeroAsset() (gas: 616271)
[PASS] testInitializeRevertsOnZeroVault() (gas: 616335)
[PASS] testInitializeSetsStateCorrectly() (gas: 39537)
[PASS] testVaultReturnExpectedValue(uint160) (runs: 258, : 1213012, ~: 1213012)
Suite result: ok. 12 passed; 0 failed; 0 skipped; finished in 382.16ms (369.56ms CPU time)

Ran 44 tests for test/unit/permissions/protocols/EigenLayerVerifier.t.sol:EigenLayerVerifierTest
[PASS] testInitializeCorrectlyGrantsAdminRole() (gas: 2791211)
[PASS] testInitializeCorrectlyGrantsRoles() (gas: 3078420)
[PASS] testInitializeRevertsIfCalledTwice() (gas: 20003)
[PASS] testInitializeRevertsOnZeroAdmin() (gas: 2005608)
[PASS] testInitializeRevertsOnZeroHolder() (gas: 2143355)
[PASS] testInitializeRevertsOnZeroRole() (gas: 2144968)
[PASS] testInitializeWithArrayLengthMismatchMoreHolders() (gas: 2286061)
[PASS] testVerifyCallCompleteQueuedWithdrawal(uint256) (runs: 258, : 44664, ~: 44664)
[PASS] testVerifyCallCompleteQueuedWithdrawalRevertsOnInvalidStaker() (gas: 37373)
[PASS] testVerifyCallCompleteQueuedWithdrawalRevertsOnInvalidStrategy() (gas: 39823)
[PASS] testVerifyCallCompleteQueuedWithdrawalRevertsOnInvalidToken() (gas: 42213)
[PASS] testVerifyCallCompleteQueuedWithdrawalRevertsOnInvalidTokensLength() (gas: 36212)
[PASS] testVerifyCallCompleteQueuedWithdrawalRevertsOnInvalidWithdrawalStrategiesLength() (gas: 38659)
[PASS] testVerifyCallCompleteQueuedWithdrawalRevertsOnMalformedCallData() (gas: 44794)
[PASS] testVerifyCallCompleteQueuedWithdrawalRevertsOnReceiveAsTokensFalse() (gas: 35798)
[PASS] testVerifyCallCompleteQueuedWithdrawalRevertsOnZeroWithdrawalStrategyAddress() (gas: 36061)
[PASS] testVerifyCallDelegateTo() (gas: 28990)
[PASS] testVerifyCallDelegateToRevertsOnInvalidOperator() (gas: 27415)
[PASS] testVerifyCallDelegateToRevertsOnMalformedCallData() (gas: 29129)
[PASS] testVerifyCallDepositIntoStrategy(uint256) (runs: 258, : 32375, ~: 32375)
[PASS] testVerifyCallDepositIntoStrategyRevertsOnInvalidAsset() (gas: 30934)
[PASS] testVerifyCallDepositIntoStrategyRevertsOnInvalidStrategy() (gas: 28565)
[PASS] testVerifyCallDepositIntoStrategyRevertsOnMalformedCallData() (gas: 32276)
[PASS] testVerifyCallDepositIntoStrategyRevertsOnZeroShares() (gas: 31525)
[PASS] testVerifyCallIgnoresVerificationData(uint256) (runs: 258, : 37965, ~: 37965)
[PASS] testVerifyCallProcessClaim(uint256) (runs: 258, : 42425, ~: 42425)
[PASS] testVerifyCallProcessClaimRevertsOnInvalidEarner() (gas: 37077)
[PASS] testVerifyCallProcessClaimRevertsOnInvalidReceiver() (gas: 39413)
[PASS] testVerifyCallProcessClaimRevertsOnMalformedCallData() (gas: 42478)
[PASS] testVerifyCallQueueWithdrawals(uint256) (runs: 258, : 32968, ~: 32968)
[PASS] testVerifyCallQueueWithdrawalsRevertsOnInvalidDepositSharesLength() (gas: 28840)
[PASS] testVerifyCallQueueWithdrawalsRevertsOnInvalidParamsLength() (gas: 31504)
[PASS] testVerifyCallQueueWithdrawalsRevertsOnInvalidStrategiesLength() (gas: 29028)
[PASS] testVerifyCallQueueWithdrawalsRevertsOnInvalidStrategy() (gas: 30288)
[PASS] testVerifyCallQueueWithdrawalsRevertsOnMalformedCallData() (gas: 32662)
[PASS] testVerifyCallQueueWithdrawalsRevertsOnZeroDepositShares() (gas: 28683)
[PASS] testVerifyCallQueueWithdrawalsRevertsOnZeroStrategyAddress() (gas: 26458)
[PASS] testVerifyCallRevertsOnInsufficientCallDataLength() (gas: 19229)
[PASS] testVerifyCallRevertsOnNonZeroValue() (gas: 23769)
```

```
[PASS] testVerifyCallRevertsOnUnauthorizedCaller() (gas: 25374)
[PASS] testVerifyCallRevertsOnUnknownContract() (gas: 25787)
[PASS] testVerifyCallRevertsOnUnknownSelectorDelegationManager() (gas: 22313)
[PASS] testVerifyCallRevertsOnUnknownSelectorRewardsCoordinator() (gas: 22268)
[PASS] testVerifyCallRevertsOnUnknownSelectorStrategyManager() (gas: 22192)
Suite result: ok. 44 passed; 0 failed; 0 skipped; finished in 405.50ms (402.49ms CPU time)

Ran 4 tests for test/unit/factories/Factory.t.sol:FactoryTest
[PASS] testBlackList() (gas: 3781818)
[PASS] testCreate() (gas: 2171665)
[PASS] testCreateEntity() (gas: 2859942)
[PASS] testProposeAndAcceptImplementation() (gas: 2510577)
Suite result: ok. 4 passed; 0 failed; 0 skipped; finished in 7.68ms (5.82ms CPU time)

Ran 28 tests for test/unit/queues/DepositQueue2.t.sol:DepositQueueTest2
[PASS] testAssetReturnExpectedValue() (gas: 17735)
[PASS] testCanBeRemoved() (gas: 288673)
[PASS] testCancelDepositRequestRevertsOnClaimable() (gas: 319601)
[PASS] testCancelDepositRequestRevertsOnNoPendingRequest() (gas: 300240)
[PASS] testCancelDepositRequestSuccess() (gas: 222085)
[PASS] testClaimReturnsFalseIfNotClaimable() (gas: 224788)
[PASS] testClaimSuccess() (gas: 287862)
[PASS] testClaimSuccess_MultipleClaimsShouldNotThrow() (gas: 289552)
[PASS] testClaimableOfReturnsCorrectValuesForProcessedDeposit() (gas: 316599)
[PASS] testClaimableOfReturnsZeroWithoutDeposit() (gas: 17937)
[PASS] testClaimableOfReturnsZeroWithoutProcessing() (gas: 221505)
[PASS] testConstructorSetsUniqueStorageSlotsForDepositQueue() (gas: 319346)
[PASS] testConstructorSetsUniqueStorageSlotsForParentQueue() (gas: 13205)
[PASS] testDepositRevertsOnZeroAssets() (gas: 19141)
[PASS] testDepositRevertsWhenDepositNotAllowed() (gas: 228634)
[PASS] testDepositRevertsWhenPendingRequestExists() (gas: 236556)
[PASS] testDepositRevertsWhenQueuePaused() (gas: 49918)
[PASS] testDepositSuccess(uint224) (runs: 258, : 226088, ~: 226088)
[PASS] testHandleReportCorrectlyHandlesFees() (gas: 469928)
[PASS] testHandleReportCorrectlyHandlesFeesWithMultipleDepositors() (gas: 503736)
[PASS] testHandleReportCorrectlyHandlesFirstDepositAfterQueueCreation() (gas: 377568)
[PASS] testHandleReportCorrectlyHandlesMultipleDeposits(uint32,uint32) (runs: 257, : 401316, ~: 401316)
[PASS] testHandleReportProcessesValidBatch() (gas: 370216)
[PASS] testHandleReportSkipsWhenNoEligibleRequests() (gas: 234227)
[PASS] testRequestOfReturnsCorrectValuesForPendingRequest() (gas: 219663)
[PASS] testRequestOfReturnsEmptyForNewUser() (gas: 18485)
[PASS] testUnclaimedRequests() (gas: 381748)
[PASS] testVaultReturnExpectedValue() (gas: 17806)
Suite result: ok. 28 passed; 0 failed; 0 skipped; finished in 562.10ms (559.75ms CPU time)

Ran 8 tests for test/unit/vaults/Subvault.t.sol:SubvaultTest
[PASS] testConstructorSetsUniqueVaultModuleSlot() (gas: 1384403)
[PASS] testConstructorSetsUniqueVerifierModuleSlot() (gas: 1384479)
[PASS] testInitializeEmitsInitializedEvent() (gas: 1380597)
[PASS] testInitializeRevertsIfCalledTwice() (gas: 1381003)
[PASS] testInitializeRevertsOnInsufficientParameters() (gas: 823911)
[PASS] testInitializeRevertsOnMalformedParameters() (gas: 822348)
[PASS] testInitializeSetsVault() (gas: 1380288)
[PASS] testInitializeSetsVerifier() (gas: 1380247)
Suite result: ok. 8 passed; 0 failed; 0 skipped; finished in 4.55ms (3.80ms CPU time)

Ran 2 tests for test/unit/modules/SubvaultModule.t.sol:SubvaultModuleTest
[PASS] testCreate() (gas: 1078520)
[PASS] testPullAssets() (gas: 1391037)
Suite result: ok. 2 passed; 0 failed; 0 skipped; finished in 3.76ms (2.54ms CPU time)

Ran 23 tests for test/unit/permissions/protocols/ERC20Verifier.t.sol:ERC20VerifierTest
[PASS] testInitializeCorrectlyGrantsAdminRole() (gas: 1910503)
[PASS] testInitializeCorrectlyGrantsRoles() (gas: 2197714)
[PASS] testInitializeRevertsIfCalledTwice() (gas: 19838)
[PASS] testInitializeRevertsOnZeroAdmin() (gas: 1124922)
[PASS] testInitializeRevertsOnZeroHolder() (gas: 1262692)
[PASS] testInitializeRevertsOnZeroRole() (gas: 1264348)
[PASS] testInitializeWithArrayLengthMismatchMoreHolders() (gas: 1405425)
[PASS] testVerifyCallApprove(uint256) (runs: 258, : 29766, ~: 29766)
[PASS] testVerifyCallApproveRevertsOnInvalidRecipient() (gas: 28528)
```

```
[PASS] testVerifyCallApproveRevertsOnMalformedCallData() (gas: 21609)
[PASS] testVerifyCallApproveRevertsOnZeroRecipient() (gas: 24792)
[PASS] testVerifyCallApproveWithZeroAmount() (gas: 29700)
[PASS] testVerifyCallIgnoresVerificationData(uint256) (runs: 258, : 35370, ~: 35370)
[PASS] testVerifyCallRevertsOnInsufficientCallData() (gas: 21368)
[PASS] testVerifyCallRevertsOnInvalidAsset() (gas: 23176)
[PASS] testVerifyCallRevertsOnNonZeroValue() (gas: 21557)
[PASS] testVerifyCallRevertsOnUnauthorizedCaller() (gas: 25515)
[PASS] testVerifyCallRevertsOnUnknownSelector() (gas: 26571)
[PASS] testVerifyCallTransfer(uint256) (runs: 258, : 30211, ~: 30211)
[PASS] testVerifyCallTransferRevertsOnInvalidRecipient() (gas: 26216)
[PASS] testVerifyCallTransferRevertsOnMalformedCallData() (gas: 21632)
[PASS] testVerifyCallTransferRevertsOnZeroAmount() (gas: 26929)
[PASS] testVerifyCallTransferRevertsOnZeroRecipient() (gas: 24830)
Suite result: ok. 23 passed; 0 failed; 0 skipped; finished in 176.64ms (175.17ms CPU time)

Ran 15 tests for test/unit/managers/RiskManager.t.sol:RiskManagerTest
[PASS] test() (gas: 207)
[PASS] testAllowAssets() (gas: 37484827)
[PASS] testConvertToShares() (gas: 36563325)
[PASS] testConvertToShares_RevertOnOverflow(bool) (runs: 258, : 36546974, ~: 36547027)
[PASS] testCreate() (gas: 37332151)
[PASS] testDisallowAssets() (gas: 37338279)
[PASS] testMaxDeposit() (gas: 37599590)
[PASS] testModify() (gas: 38765301)
[PASS] testModifyPendingAssets_NotRevertOnLimitExceededWhenRemoving() (gas: 37775781)
[PASS] testModifyPendingAssets_RevertOnLimitExceededWhenAdding() (gas: 37657677)
[PASS] testModifySubvaultBalance_NotRevertOnLimitExceededWhenRemoving() (gas: 37579127)
[PASS] testModifySubvaultBalance_RevertOnLimitExceededWhenAdding() (gas: 37567641)
[PASS] testModifyVaultBalance_NotRevertOnLimitExceededWhenRemoving() (gas: 37620168)
[PASS] testModifyVaultBalance_RevertOnLimitExceededWhenAdding() (gas: 37612296)
[PASS] testSetVault() (gas: 36461093)
Suite result: ok. 15 passed; 0 failed; 0 skipped; finished in 2.38s (2.38s CPU time)

Ran 13 tests for test/unit/managers/ShareManager.t.sol:ShareManagerTest
[PASS] test() (gas: 207)
[PASS] testAccountInfo() (gas: 35950109)
[PASS] testAllocateShares() (gas: 37823154)
[PASS] testClaimShares() (gas: 35978025)
[PASS] testCreate() (gas: 35933201)
[PASS] testFlags() (gas: 35956096)
[PASS] testIsDepositorWhitelisted() (gas: 35989536)
[PASS] testMintBurn() (gas: 36966867)
[PASS] testSetVault() (gas: 35919887)
[PASS] testTargetLockup() (gas: 35979844)
[PASS] testUpdateChecks() (gas: 36056733)
[PASS] testUpdateChecks_blacklist() (gas: 36031999)
[PASS] testUpdateChecks_transferWhitelist() (gas: 36031213)
Suite result: ok. 13 passed; 0 failed; 0 skipped; finished in 157.41ms (144.78ms CPU time)

Ran 31 tests for test/unit/queues/RedeemQueue2.t.sol:RedeemQueueTest2
[PASS] testAssetReturnExpectedValue() (gas: 17646)
[PASS] testBatchAtReturnsZeroForInvalidIndex(uint32) (runs: 258, : 16294, ~: 16294)
[PASS] testCanBeRemoved() (gas: 376491)
[PASS] testClaimReturnsAssetByThePriceOfTheNextReport() (gas: 396257)
[PASS] testClaimReturnsZero_BatchNotHandled() (gas: 370117)
[PASS] testClaimReturnsZero_NoReport() (gas: 21055)
[PASS] testClaimReturnsZero_NoUserRequestButReportsExist() (gas: 364350)
[PASS] testClaimReturnsZero_RequestNotIncludedInReport() (gas: 487519)
[PASS] testClaimSuccess() (gas: 534882)
[PASS] testConstructorSetsUniqueStorageSlotsForParentQueue() (gas: 13117)
[PASS] testConstructorSetsUniqueStorageSlotsForRedeemQueue() (gas: 55233)
[PASS] testFullRedeemFlow_MultipleUsers_DifferentBatches() (gas: 4177882)
[PASS] testFullRedeemFlow_MultipleUsers_DifferentReports() (gas: 831950)
[PASS] testFullRedeemFlow_MultipleUsers_SameReport() (gas: 615148)
[PASS] testGetStateReturnsInitialValues() (gas: 24329)
[PASS] testHandleBatchesEarlyExitWhenNoBatches() (gas: 363934)
[PASS] testHandleBatchesHandlesPartialLiquidity() (gas: 593583)
[PASS] testHandleBatchesProcessesBatchCorrectly() (gas: 378978)
[PASS] testHandleReportCorrectlyHandlesFirstRedeemAfterQueueCreation() (gas: 478984)
```

```
[PASS] testHandleReportCreatesBatchAndIncreasesDemandAssets() (gas: 350441)
[PASS] testHandleReportCreatesBatchAndIncreasesDemandAssets_WithMultipleRequests() (gas: 361499)
[PASS] testHandleReportDoesNotProcessIfNoRequestsWereMade() (gas: 63742)
[PASS] testRedeemCorrectlyHandlesFees() (gas: 376497)
[PASS] testRedeemRevertsOnZeroShares() (gas: 20993)
[PASS] testRedeemRevertsWhenQueuePaused() (gas: 51712)
[PASS] testRedeemSuccess() (gas: 203593)
[PASS] testRequestsOfReturnsCorrectValuesForMultipleRequests() (gas: 202090)
[PASS] testRequestsOfReturnsCorrectValuesForMultipleRequests_WithDifferentTimestamps() (gas: 320229)
[PASS] testRequestsOfReturnsCorrectValuesForPendingRequest() (gas: 191095)
[PASS] testRequestsOfReturnsEmptyForNewUser() (gas: 18678)
[PASS] testVaultReturnExpectedValue() (gas: 17806)
Suite result: ok. 31 passed; 0 failed; 0 skipped; finished in 66.38ms (64.10ms CPU time)

Ran 6 tests for test/unit/queues/SignatureDepositQueue.t.sol:SignatureDepositQueueTest
[PASS] test() (gas: 207)
[PASS] testCreate() (gas: 6403772)
[PASS] testDeposit() (gas: 40306223)
[PASS] testDepositETH() (gas: 40158639)
[PASS] testFuzzDeposit(int16[10]) (runs: 258, : 42848964, ~: 42844470)
[PASS] testFuzzDepositCallerRecipient(address,address) (runs: 258, : 40217170, ~: 40217170)
Suite result: ok. 6 passed; 0 failed; 0 skipped; finished in 30.29s (36.31s CPU time)

Ran 11 tests for test/unit/oracles/Oracle.t.sol:OracleTest
[PASS] test() (gas: 229)
[PASS] testAddAndRemoveSupportedAsset() (gas: 39119235)
[PASS] testCreate() (gas: 36477215)
[PASS] testFuzzMultipleAssets(int16[],int16[]) (runs: 257, : 52628962, ~: 49947984)
[PASS] testFuzzMultipleAssetsWithInvalidSubmits(int16[],int16[]) (runs: 257, : 47287198, ~: 45087100)
[PASS] testFuzzNonSuspiciousDeviation(uint8) (runs: 258, : 38707663, ~: 37860162)
[PASS] testFuzzSuspiciousDeviation(int16[]) (runs: 257, : 38143395, ~: 38050439)
[PASS] testHandleReport() (gas: 36642491)
[PASS] testSecurityParams() (gas: 36521910)
[PASS] testSetVault() (gas: 36461161)
[PASS] testValidatePrice() (gas: 109524765)
Suite result: ok. 11 passed; 0 failed; 0 skipped; finished in 39.09s (80.07s CPU time)

Ran 6 tests for test/unit/libraries/FenwickTreeLibrary.t.sol:Unit
[PASS] testDiff() (gas: 19772260)
[PASS] testExtend() (gas: 346395)
[PASS] testFenwickTreeGetGasUsage() (gas: 34888894964)
[PASS] testFenwickTreeModifyGasUsage() (gas: 131528514)
[PASS] testInitialize() (gas: 1451140)
[PASS] testModifyAndGet() (gas: 525616)
Suite result: ok. 6 passed; 0 failed; 0 skipped; finished in 45.62s (45.69s CPU time)

Ran 28 tests for test/unit/permissions/Consensus.t.sol:ConsensusTest
[PASS] test() (gas: 229)
[PASS] testAddSignerAndThreshold() (gas: 1802243)
[PASS] testAddSignerInvalidType() (gas: 1819956)
[PASS] testAddSignerTwiceFails() (gas: 1800105)
[PASS] testAddSignerZeroAddress() (gas: 1700882)
[PASS] testCheckEmptySignatures() (gas: 1806094)
[PASS] testCheckInvalidSigner() (gas: 1807749)
[PASS] testCheckSignatures_EIP1271_Invalid() (gas: 1993604)
[PASS] testCheckSignatures_EIP1271_Invalid_DuplicateSignatures(uint128,uint128) (runs: 258, : 2278307, ~: 2278307)
[PASS] testCheckSignatures_EIP1271_Invalid_WrongOrder(uint128,uint128) (runs: 258, : 2278698, ~: 2278698)
[PASS] testCheckSignatures_EIP1271_MultipleValid(uint128,uint128) (runs: 258, : 2281223, ~: 2281223)
[PASS] testCheckSignatures_EIP1271_Valid() (gas: 2012950)
[PASS] testCheckSignatures_EIP712_Invalid() (gas: 1816546)
[PASS] testCheckSignatures_EIP712_Invalid_DuplicateSignatures(uint128,uint128) (runs: 258, : 1874056, ~: 1874056)
[PASS] testCheckSignatures_EIP712_Invalid_WrongOrder(uint128,uint128) (runs: 257, : 1875341, ~: 1875341)
[PASS] testCheckSignatures_EIP712_MultipleValid(uint128,uint128) (runs: 257, : 1883139, ~: 1883139)
[PASS] testCheckSignatures_EIP712_NotEnoughSignatures() (gas: 1858458)
[PASS] testCheckSignatures_EIP712_Valid() (gas: 1814503)
[PASS] testFuzzCheckSignaturesDuplicates(bool[],uint8) (runs: 256, : 33706431, ~: 32979653)
[PASS] testFuzzCheckSignaturesInvalidOrder(bool[],uint8,uint8) (runs: 256, : 36194219, ~: 37737060)
[PASS] testFuzzCheckSignaturesInvalidSignature(bool[],uint8) (runs: 256, : 34256119, ~: 34116101)
[PASS] testFuzzSignaturesValid(bool[]) (runs: 257, : 11121482, ~: 11597839)
[PASS] testInitializeOwner() (gas: 1700083)
```

```
[PASS] testRemoveNonexistentSigner() (gas: 1825787)
[PASS] testRemoveSigner() (gas: 1855361)
[PASS] testSetThresholdInvalid() (gas: 1995209)
[PASS] testSignerAtAndLength() (gas: 1799848)
[PASS] testSignerAtOutOfBounds() (gas: 1803034)
Suite result: ok. 28 passed; 0 failed; 0 skipped; finished in 53.26s (141.24s CPU time)

Ran 5 tests for test/unit/queues/SignatureRedeemQueue.t.sol:SignatureRedeemQueueTest
[PASS] test() (gas: 207)
[PASS] testCreate() (gas: 6427460)
[PASS] testFuzzRedeemCallerRecipient(address[64]) (runs: 258, : 57522713, ~: 57529435)
[PASS] testRedeem() (gas: 40298556)
[PASS] testRedeemETH() (gas: 41256353)
Suite result: ok. 5 passed; 0 failed; 0 skipped; finished in 111.64s (111.72s CPU time)

Ran 11 tests for test/unit/queues/RedeemQueue.t.sol:RedeemQueueTest
[PASS] test() (gas: 229)
[PASS] testCreate() (gas: 37558754)
[PASS] testFuzzMultipleBatches(uint8[],uint8) (runs: 257, : 179952623, ~: 186536965)
[PASS] testFuzzMultipleRedeemQueues(int16[10]) (runs: 258, : 50190838, ~: 50190838)
[PASS] testFuzzMultipleRedeemQueuesDifferentAssets(int16[10]) (runs: 258, : 56208617, ~: 56208617)
[PASS] testFuzzRedeem(uint16[200],int16[200]) (runs: 258, : 69037038, ~: 74742408)
[PASS] testRedeemETH() (gas: 39353198)
[PASS] testRedeemFee() (gas: 39170826)
[PASS] testRedeemInterval() (gas: 39428338)
[PASS] testRedeemInterval_Claimable() (gas: 39521494)
[PASS] testSingleDepositRedeem() (gas: 39290271)
Suite result: ok. 11 passed; 0 failed; 0 skipped; finished in 115.88s (176.31s CPU time)

Ran 21 tests for test/unit/queues/DepositQueue.t.sol:DepositQueueTest
[PASS] test() (gas: 207)
[PASS] testCancelSingleDepositRequest() (gas: 38038004)
[PASS] testClaim() (gas: 37969864)
[PASS] testDeposit() (gas: 37940021)
[PASS] testDepositETH() (gas: 37876386)
[PASS] testDepositInterval() (gas: 38160665)
[PASS] testDepositInterval_Claimable() (gas: 38324294)
[PASS] testDepositLimitExceeded() (gas: 38109847)
[PASS] testDepositNotAllowed() (gas: 37857073)
[PASS] testDepositPausedQueue() (gas: 37715792)
[PASS] testDepositRequestAmountIntegrity(uint8,uint8,uint8) (runs: 256, : 40764848, ~: 40015214)
[PASS] testFuzzCancelMultipleDepositRequests(int16,int16[256],bool[256]) (runs: 258, : 69070958, ~: 69072929)
[PASS] testFuzzDepositMultipleQueuesMultipleAssets(int16[10],int16[10]) (runs: 258, : 55579002, ~: 55574132)
[PASS] testFuzzDepositMultipleQueuesSingleAsset(int16[100],uint16,int16) (runs: 258, : 155315002, ~: 155315985)
[PASS] testFuzzDepositsMultipleUsers(int16[256],int16[256]) (runs: 258, : 122417659, ~: 121159144)
[PASS] testFuzzDepositsOneUser(int16[100],int16[100]) (runs: 258, : 62796238, ~: 62367443)
[PASS] testHandleReport() (gas: 38279232)
[PASS] testMintFees() (gas: 38221373)
[PASS] testMultipleDeposit() (gas: 38984535)
[PASS] testRemoveQueue() (gas: 38045471)
[PASS] testRequestsExtend() (gas: 39713345)
Suite result: ok. 21 passed; 0 failed; 0 skipped; finished in 116.48s (227.52s CPU time)

Ran 52 test suites in 117.71s (517.65s CPU time): 430 tests passed, 0 failed, 0 skipped (430 total tests)
Done in 120.14s.
```

9 About Nethermind

Nethermind is a Blockchain Research and Software Engineering company. Our work touches every part of the web3 ecosystem - from layer 1 and layer 2 engineering, cryptography research, and security to application-layer protocol development. We offer strategic support to our institutional and enterprise partners across the blockchain, digital assets, and DeFi sectors, guiding them through all stages of the research and development process, from initial concepts to successful implementation.

We offer security audits of projects built on EVM-compatible chains and Starknet. We are active builders of the Starknet ecosystem, delivering a node implementation, a block explorer, a Solidity-to-Cairo transpiler, and formal verification tooling. Nethermind also provides strategic support to our institutional and enterprise partners in blockchain, digital assets, and decentralized finance (DeFi). In the next paragraphs, we introduce the company in more detail.

Blockchain Security: At Nethermind, we believe security is vital to the health and longevity of the entire Web3 ecosystem. We provide security services related to Smart Contract Audits, Formal Verification, and Real-Time Monitoring. Our Security Team comprises blockchain security experts in each field, often collaborating to produce comprehensive and robust security solutions. The team has a strong academic background, can apply state-of-the-art techniques, and is experienced in analyzing cutting-edge Solidity and Cairo smart contracts, such as ArgentX and StarkGate (the bridge connecting Ethereum and StarkNet). Most team members hold a Ph.D. degree and actively participate in the research community, accounting for 240+ articles published and 1,450+ citations in Google Scholar. The security team adopts customer-oriented and interactive processes where clients are involved in all stages of the work.

Blockchain Core Development: Our core engineering team, consisting of over 20 developers, maintains, improves, and upgrades our flagship product - the Nethermind Ethereum Execution Client. The client has been successfully operating for several years, supporting both the Ethereum Mainnet and its testnets, and now accounts for nearly a quarter of all synced Mainnet nodes. Our unwavering commitment to Ethereum's growth and stability extends to sidechains and layer 2 solutions. Notably, we were the sole execution layer client to facilitate Gnosis Chain's Merge, transitioning from Aura to Proof of Stake (PoS), and we are actively developing a full-node client to bolster Starknet's decentralization efforts. Our core team equips partners with tools for seamless node set-up, using generated docker-compose scripts tailored to their chosen execution client and preferred configurations for various network types.

DevOps and Infrastructure Management: Our infrastructure team ensures our partners' systems operate securely, reliably, and efficiently. We provide infrastructure design, deployment, monitoring, maintenance, and troubleshooting support, allowing you to focus on your core business operations. Boasting extensive expertise in Blockchain as a Service, private blockchain implementations, and node management, our infrastructure and DevOps engineers are proficient with major cloud solution providers and can host applications in-house or on clients' premises. Our global in-house SRE teams offer 24/7 monitoring and alerts for both infrastructure and application levels. We manage over 5,000 public and private validators and maintain nodes on major public blockchains such as Polygon, Gnosis, Solana, Cosmos, Near, Avalanche, Polkadot, Aptos, and StarkWare L2. Sedge is an open-source tool developed by our infrastructure experts, designed to simplify the complex process of setting up a proof-of-stake (PoS) network or chain validator. Sedge generates docker-compose scripts for the entire validator set-up based on the chosen client, making the process easier and quicker while following best practices to avoid downtime and being slashed.

Cryptography Research: At Nethermind, our cryptography Research team conducts cutting-edge internal research and collaborates closely with external partners on cryptographic protocols, consensus design, succinct arguments and folding schemes, elliptic curve-based STARK protocols, post-quantum security and zero-knowledge proofs (ZKPs). Our research has led to influential contributions, including Zinc (Crypto '25), Mova, FLI (Asiacrypt '24), and foundational results in Fiat-Shamir security and STARK proof batching. Complementing this theoretical work, our engineering expertise is demonstrated through implementations such as the Latticefold aggregation scheme, the Labrador proof system, zkvm-benchmarks, and Plonk Verifier in Cairo. This combined strength in theory and engineering enables us to deliver cutting-edge cryptographic solutions to partners and clients.

Smart Contract Development & DeFi Research: Our smart contract development and DeFi research team comprises 40+ world-class engineers who collaborate closely with partners to identify needs and work on value-adding projects. The team specializes in Solidity and Cairo development, architecture design, and DeFi solutions, including DEXs, AMMs, structured products, derivatives, and money market protocols, as well as ERC20, 721, and 1155 token design. Our research and data analytics focuses on three key areas: technical due diligence, market research, and DeFi research. Utilizing a data-driven approach, we offer in-depth insights and outlooks on various industry themes.

Our suite of L2 tooling: Warp is Starknet's approach to EVM compatibility. It allows developers to take their Solidity smart contracts and transpile them to Cairo, Starknet's smart contract language. In the short time since its inception, the project has accomplished many achievements, including successfully transpiling Uniswap v3 onto Starknet using Warp.

- **Voyager** is a user-friendly Starknet block explorer that offers comprehensive insights into the Starknet network. With its intuitive interface and powerful features, Voyager allows users to easily search for and examine transactions, addresses, and contract details. As an essential tool for navigating the Starknet ecosystem, Voyager is the go-to solution for users seeking in-depth information and analysis;
- **Horus** is an open-source formal verification tool for StarkNet smart contracts. It simplifies the process of formally verifying Starknet smart contracts, allowing developers to express various assertions about the behavior of their code using a simple assertion language;
- **Juno** is a full-node client implementation for Starknet, drawing on the expertise gained from developing the Nethermind Client. Written in Golang and open-sourced from the outset, Juno verifies the validity of the data received from Starknet by comparing it to proofs retrieved from Ethereum, thus maintaining the integrity and security of the entire ecosystem.

General Advisory to Clients

As auditors, we recommend that any changes or updates made to the audited codebase undergo a re-audit or security review to address potential vulnerabilities or risks introduced by the modifications. By conducting a re-audit or security review of the modified codebase, you can significantly enhance the overall security of your system and reduce the likelihood of exploitation. However, we do not possess the authority or right to impose obligations or restrictions on our clients regarding codebase updates, modifications, or subsequent audits. Accordingly, the decision to seek a re-audit or security review lies solely with you.

Disclaimer

This report is based on the scope of materials and documentation provided by you to Nethermind in order that Nethermind could conduct the security review outlined in **1. Executive Summary** and **2. Audited Files**. The results set out in this report may not be complete nor inclusive of all vulnerabilities. Nethermind has provided the review and this report on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. Blockchain technology remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. This report does not indicate the endorsement of any particular project or team, nor guarantee its security. No third party should rely on this report in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. To the fullest extent permitted by law, Nethermind disclaims any liability in connection with this report, its content, and any related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. Nethermind does not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and Nethermind will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate. FOR AVOIDANCE OF DOUBT, THE 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 OTHER ADVICE.