Security Review Report
NM-0485 Royco

NETHERMIND
SECURITY

(April 17, 2025)

Contents

1 Executive Summary

This document presents the security review performed by Nethermind Security for Royco contracts. The audit review focused on Royco V2 protocol which has one major component that contains the core logic and connects the system entirely namely; Incentive locker.

The audit comprises 919 lines of solidity code. **The audit was performed using** (a) manual analysis of the codebase, (b) automated analysis tools, and (c) creation of test cases.

Along this document, we report 3 points of attention, where they are classified as two `Low` and one `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 compilation, tests, and automated tests. Section 9 concludes the document.

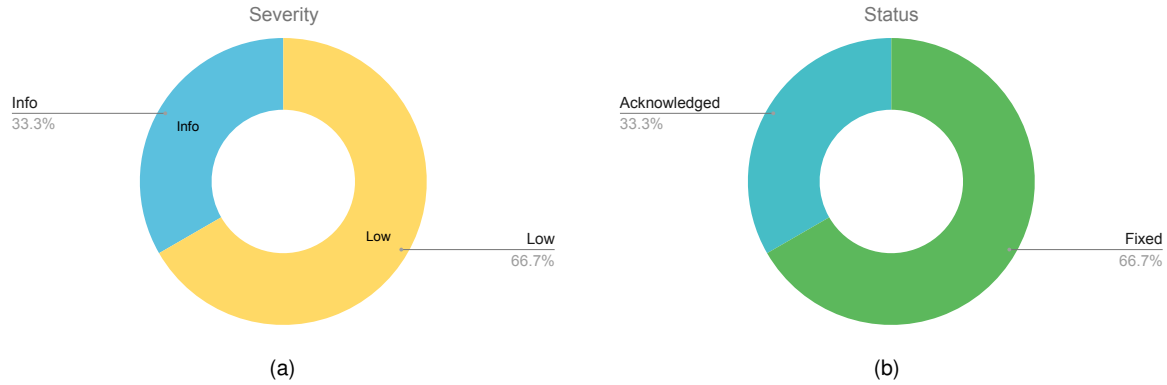


Fig. 1: Distribution of issues: Critical (0), **High** (0), **Medium** (0), **Low** (2), **Undetermined** (0), **Informational** (1), **Best Practices** (0). **Distribution of status: Fixed** (2), **Acknowledged** (1), **Mitigated** (0), **Unresolved** (0)

Summary of the Audit

Audit Type	Security Review
Final Report	April 17, 2025
Repositories	royco-v2
Initial Commit	1716b71
Final Commit	4d89987
Documentation	Provided in the code's natspec
Documentation Assessment	Medium
Test Suite Assessment	High

2 Audited Files

	Contract	LoC	Comments	Ratio	Blank	Total
1	src/interfaces/IActionVerifier.sol	41	32	78.0%	5	78
2	src/interfaces/IERC20.sol	11	59	536.4%	9	79
3	src/periphery/market-hubs/MultiplierMarketHub.sol	59	51	86.4%	23	133
4	src/core/IncentiveLocker.sol	387	246	63.6%	84	717
5	src/core/base/PointsRegistry.sol	82	92	112.2%	27	201
6	src/core/action-verifiers/uma/UmaMerkleChefAV.sol	192	130	67.7%	46	368
7	src/core/action-verifiers/uma/base/UmaMerkleOracleBase.sol	147	116	78.9%	39	302
	Total	**919**	**726**	**79.0%**	**233**	**1878**

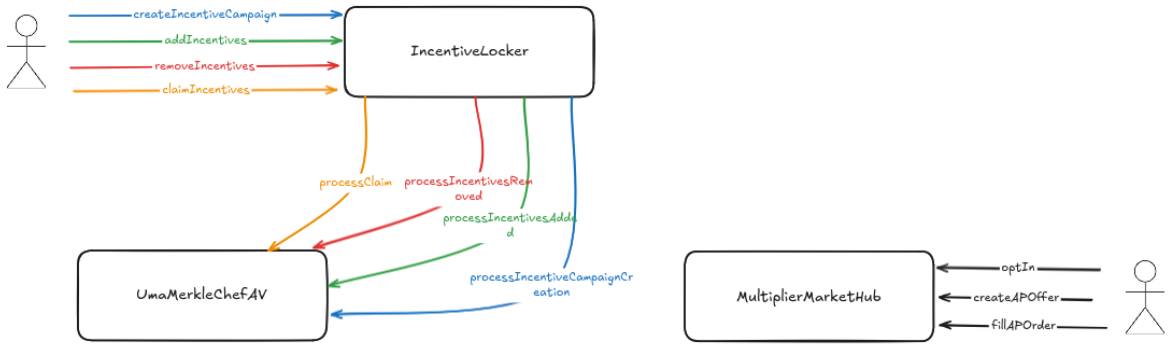
3 Summary of Issues

	Finding	Severity	Update
1	Duplicate entries in `_modifyIncentiveStreams(...)` result in loss of incentives	Low	Fixed
2	Malicious incentive token can restrict claiming of other incentives	Low	Acknowledged
3	`UmaMerkleOracleBase::blacklistAsserters(...)` does not revoke assertion privileges	Info	Fixed

4 System Overview

Royco V2 introduces Incentivized Action Markets (IAMs), enabling anyone to set up a market that rewards specific actions, whether on-chain or off-chain. Within these markets, Incentive Providers (IPs) propose rewards—such as tokens or points—for Action Providers (APs) willing to complete certain tasks. IPs and APs negotiate terms by exchanging offers until they settle on a mutually acceptable reward. Once an agreement is reached, the AP is expected to carry out the agreed-upon actions as outlined in the on-chain contract to receive the promised incentive.

The protocol is composed of three main contracts with the `IncentiveLocker` being the main contract that contains the core logic.



The `IncentiveLocker` is the central contract to manage incentive campaigns. Incentive Providers (IPs) use it to deposit tokens or points into campaigns that incentivize specific actions. The main flows include the following:

- **Campaign Creation**: IPs create campaigns by specifying an action verifier and incentive details.
- **Incentive Management**: IPs and co-IPs can add incentive tokens during the campaign, while only IPs can remove them.
- **Incentives Claims**: Action Providers (APs) claim rewards by proving action completion (verified by `ActionVerifier` hooks).

`UmaMerkleChefAV` is an action verifier used by the `IncentiveLocker` to validate and process incentive claims based on Merkle proofs. It enables streamed rewards using UMA's optimistic oracle. The main flows include:

- **Merkle Root Publishing**: An off-chain oracle periodically posts Merkle roots containing AP reward data.
- **Stream Adjustment**: Incentive emission rates can be increased or decreased over time upon adding or removing incentives, respectively.
- **Merkle Proofs Attestations**: Whitelisted attesters or IPs submit Merkle proofs of owed amounts, which are verified and confirmed or disputed by the off-chain oracle.

The `MultiplierMarketHub` enables negotiation and opt-in between IPs and APs for multiplier-based campaigns, and the contract is used off-chain for rewards calculations.

- APs express interest in a campaign by opting in.
- APs submit counter-offers that include a reward multiplier and optional parameters (e.g., task size).
- IPs can accept AP offers to finalize agreements, which signals readiness for the AP to perform the incentivized action.

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 [Low] Duplicate entries in `_modifyIncentiveStreams(...)` result in loss of incentives

File(s): UmaMerkleChefAV.sol

Description: The `IncentiveLocker::createIncentiveCampaign(...)` function is used to create a campaign which invokes `UmaMerkleChefAV::processIncentiveCampaignCreation(...)`. The `processIncentiveCampaignCreation(...)` invokes `_modifyIncentiveStreams(...)` to update the current rate for each incentive in `incentiveCampaignIdToIncentiveToCurrentRate` mapping.

```
    function processIncentiveCampaignCreation(
        bytes32 _incentiveCampaignId,
        address[] memory _incentivesOffered,
        uint256[] memory _incentiveAmountsOffered,
        bytes memory _actionParams,
        address /*_ip*/
    )
        external
        override
        onlyIncentiveLocker
    {
        ActionParams memory params = abi.decode(_actionParams, (ActionParams));
        uint32 startTimestamp = params.startTimestamp;
        uint32 endTimestamp = params.endTimestamp;

        // ...

        // Apply the modification to streams
        _modifyIncentiveStreams(Modifications.INIT_STREAMS, _incentiveCampaignId, startTimestamp, endTimestamp,
        ↪  _incentivesOffered, _incentiveAmountsOffered);
    }

    function _modifyIncentiveStreams(
        Modifications _modification,
        bytes32 _incentiveCampaignId,
        uint32 _startTimestamp,
        uint32 _endTimestamp,
        address[] memory _incentives,
        uint256[] memory _incentiveAmounts
    )
        internal
    {
        // ...

        // Make modifications to the appropriate incentive streams
        uint256 numIncentives = _incentives.length;
        uint256[] memory updatedRates = new uint256[](numIncentives);
        for (uint256 i = 0; i < numIncentives; ++i) {
            // If creating a new campaign
            if (_modification == Modifications.INIT_STREAMS) {
                // Initialize the rate on creation
                updatedRates[i] = (_incentiveAmounts[i]).divWadDown(_endTimestamp - _startTimestamp);
                // If adding or removing incentives from an already created campaign
            } else {
                // ...
            }

            // Update the rate for this incentive to the current rate
            // @audit If _incentives includes duplicate entries, it will override the rate corresponding to previous
            ↪  entry.
            incentiveCampaignIdToIncentiveToCurrentRate[_incentiveCampaignId][_incentives[i]] = updatedRates[i];
        }

        // ...
    }
```

However, if `_incentives` in `_modifyIncentiveStreams(...)` includes duplicate entries, it will override the rate corresponding to the previous entry. This would result in incorrect accounting.

Although, the merkle root submitted by `IP` can still correspond to correct accounting.

Recommendation(s): Consider handling the case where incentives array while creating a campaign includes duplicate entries.

Status: Fixed

Update from the client: Fixed in commit d273a017

6.2 [Low] Malicious incentive token can restrict claiming of other incentives

File(s): IncentiveLocker.sol

Description: The `claimIncentives(...)` function is responsible for claiming incentives from a specific incentive campaign. It uses `_remitIncentivesAndFees(...)` to process individual incentive claims by iterating over each offered incentive. To transfer incentive amount and account for protocol fee, it calls `_pushIncentivesAndAccountFees(...)`.

```
    function _remitIncentivesAndFees(
        ICS storage _ics,
        address _ap,
        address _protocolFeeClaimant,
        address[] memory _incentives,
        uint256[] memory _incentiveAmountsOwed
    )
        internal
        returns (uint256[] memory incentiveAmountsPaid, uint256[] memory protocolFeesPaid)
    {
        // Cache for gas op
        uint256 numIncentives = _incentives.length;
        // ...
        // Initialize array for event emission
        incentiveAmountsPaid = new uint256[](numIncentives);
        protocolFeesPaid = new uint256[](numIncentives);

        for (uint256 i = 0; i < numIncentives; ++i) {
            // Cache for gas op
            address incentive = _incentives[i];
            uint256 incentiveAmountOwed = _incentiveAmountsOwed[i];

            // Account for spent incentives to prevent co-mingling of incentives
            _ics.incentiveToAmountRemaining[incentive] -= incentiveAmountOwed;

            // Calculate fee amounts based on the claim ratio.
            protocolFeesPaid[i] = incentiveAmountOwed.mulWadDown(_ics.protocolFee);

            // Calculate the net incentive amount to be paid after applying fees.
            incentiveAmountsPaid[i] = incentiveAmountOwed - protocolFeesPaid[i];

            // Push incentives to the action provider and account for fees.
            // @audit If this reverts for one incentive, then the transfer for all other incentives will also revert.
            _pushIncentivesAndAccountFees(incentive, _ap, _protocolFeeClaimant, incentiveAmountsPaid[i],
             ↪  protocolFeesPaid[i]);
        }
    }

    function _pushIncentivesAndAccountFees(
        address incentive,
        address ap,
        address protocolFeeClaimant,
        uint256 incentiveAmount,
        uint256 protocolFeeAmount
    )
        internal
    {
        // Take fees and push incentives to action provider
        if (isPointsProgram(incentive)) {
            // ...
        } else {
            // Make fees claimable by fee claimant
            feeClaimantToTokenToAmount[protocolFeeClaimant][incentive] += protocolFeeAmount;
            // Transfer incentives to the action provider
            ERC20(incentive).safeTransfer(ap, incentiveAmount);
        }
    }
```

However, If `_pushIncentivesAndAccountFees(...)` reverts for one incentive, then all other incentives can't be transferred too. This exploit allows `IP` or `Co-IP` to add a malicious token as incentive which behaves maliciously after the merkle root is verified, restricting claiming of incentives.

Although, a whitelisted asserter can then add a merkle root such that it excludes the malicious token.

Recommendation(s): Consider updating the implementation for transferring the incentives to handle the revert case.

Status: Acknowledged

Update from the client: The asserters would just post another merkle root with the malicious tokens excluded from the leaves.

6.3 [Info] `UmaMerkleOracleBase::blacklistAsserters(...)` does not revoke assertion privileges

File(s): `UmaMerkleOracleBase.sol`

Description: The `blacklistAsserters(...)` function in `UmaMerkleOracleBase` is intended to revoke assertion privileges by blacklisting asserters. However, the current implementation mistakenly sets the `asserterToIsWhitelisted` mapping to `true` for the provided addresses, which has the opposite effect—whitelisting the asserters instead of blacklisting them:

```
function blacklistAsserters(address[] memory _blacklistedAsserters) public virtual onlyOwner {
    uint256 numAsserters = _blacklistedAsserters.length;
    for (uint256 i = 0; i < numAsserters; ++i) {
==>         asserterToIsWhitelisted[_blacklistedAsserters[i]] = true;
    }
    emit AssertersBlacklisted(_blacklistedAsserters);
}
```

Though malicious asserters attempting to assert an incorrect Merkle root will lose the bond amount since the oracle will dispute their assertions.

Recommendation(s): Consider setting the `asserterToIsWhitelisted` to `false` instead of `true`.

Status: Fixed.

Update from the client: Fixed in commit 3f16f86

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 Royco documentation

The Royco team has provided a comprehensive walkthrough of the project in the kick-off call, and the code natspec includes a detailed explanation of the intended functionalities. Moreover, the team addressed all questions and concerns raised by the Nethermind Security team, providing valuable insights and a comprehensive understanding of the project's technical aspects.

8 Test Suite Evaluation

8.1 Compilation Output

```
> forge compile
[] Compiling...
[] Unable to resolve imports:
      "../../../lib/solmate/src/utils/FixedPointMathLib.sol" in
       ↪ "/home/brivan/brivan/nethermind-audits/royco-v2/test/UmaMerkleChefAV/Test_RemovingIncentives_UMC.t.sol"
      "../../../lib/solmate/src/utils/FixedPointMathLib.sol" in
       ↪ "/home/brivan/brivan/nethermind-audits/royco-v2/test/UmaMerkleChefAV/Test_AddingIncentives_UMC.t.sol"
      "../../../lib/solmate/src/utils/FixedPointMathLib.sol" in
       ↪ "/home/brivan/brivan/nethermind-audits/royco-v2/test/UmaMerkleChefAV/Test_UmaMerkleOracleBase_UMC.t.sol"
with remappings:
      @openzeppelin/contracts/=/home/brivan/brivan/nethermind-audits/royco-v2/lib/openzeppelin-contracts/contracts/
      ds-test/=/home/brivan/brivan/nethermind-audits/royco-v2/lib/solmate/lib/ds-test/src/
      erc4626-tests/=/home/brivan/brivan/nethermind-audits/royco-v2/lib/openzeppelin-contracts/lib/erc4626-tests/
      forge-std/=/home/brivan/brivan/nethermind-audits/royco-v2/lib/forge-std/src/
      halmos-cheatcodes/=/home/brivan/brivan/nethermind-audits/royco-v2/lib/openzeppelin-contracts/lib/halmos-cheatcod
       ↪ es/src/
      openzeppelin-contracts/=/home/brivan/brivan/nethermind-audits/royco-v2/lib/openzeppelin-contracts/
      solmate/=/home/brivan/brivan/nethermind-audits/royco-v2/lib/solmate/src/
      uma-protocol/=/home/brivan/brivan/nethermind-audits/royco-v2/lib/uma-protocol/
[] Compiling 53 files with Solc 0.8.28
[] Solc 0.8.28 finished in 29.61s
Compiler run successful with warnings:
Warning (2072): Unused local variable.
  --> test/IncentiveLocker/Test_AddAndRemoveCoIPs.t.sol:12:9:
   |
12 |         uint256 len = bound(_coIPs.length, 1, 100);
   |         ^^^^^^^^^^

Warning (2072): Unused local variable.
  --> test/MultiplierMarketHub/Test_MultiplierMarketHub.t.sol:66:9:
   |
66 |         bytes32 apOfferHash = multiplierMarketHub.createAPOffer(incentiveCampaignId, _multiplier, _size);
   |         ^^^^^^^^^^^^^^^^^^
```

8.2 Tests Output

```
> forge test
Ran 5 tests for test/IncentiveLocker/Test_PointsRegistry.t.sol:Test_PointsRegistry
[PASS] test_PointsProgramCreation(address,string,string,uint8,uint8) (runs: 256, : 2267782, ~: 1251462)
[PASS] test_RevertIf_NotIP_UpdateSpendCaps(address,address,uint8) (runs: 256, : 1984309, ~: 861343)
[PASS] test_RevertIf_SpendingExceedsCap(address,uint8) (runs: 256, : 2179418, ~: 1229906)
[PASS] test_TransferPointsProgramOwnership(address,address) (runs: 256, : 137766, ~: 137844)
[PASS] test_UpdateSpendCaps(address,uint8) (runs: 256, : 2499725, ~: 1493416)
Suite result: ok. 5 passed; 0 failed; 0 skipped; finished in 2.32s (1.14s CPU time)

Ran 7 tests for test/MultiplierMarketHub/Test_MultiplierMarketHub.t.sol:Test_MultiplierMarketHub
[PASS] test_CreateAPOffer(address,uint96,uint256) (runs: 256, : 142732, ~: 142867)
[PASS] test_FillAPOffer(address,uint96,uint256) (runs: 256, : 147621, ~: 147849)
[PASS] test_OptIn(address) (runs: 256, : 45030, ~: 45030)
[PASS] test_RevertIf_CreateAPOfferExceedsMaxMultiplier(address,uint96,uint256) (runs: 256, : 21588, ~: 21541)
[PASS] test_RevertIf_FillAPOfferAsNotIP(address,address,uint96,uint256) (runs: 256, : 142389, ~: 142601)
[PASS] test_RevertIf_OptInMoreThanOnce(address) (runs: 256, : 43840, ~: 43840)
[PASS] test_RevertIf_OptInToNonexistantCampaign(address,bytes32) (runs: 256, : 18167, ~: 18167)
Suite result: ok. 7 passed; 0 failed; 0 skipped; finished in 4.50s (177.08ms CPU time)

Ran 2 tests for test/IncentiveLocker/Test_AddAndRemoveCoIPs.t.sol:Test_AddAndRemoveCoIPs
[PASS] test_AddCoIPs(address[]) (runs: 256, : 3634298, ~: 3661908)
[PASS] test_RemoveCoIPs(address[],uint256) (runs: 256, : 3132784, ~: 3055314)
Suite result: ok. 2 passed; 0 failed; 0 skipped; finished in 4.64s (953.91ms CPU time)

Ran 5 tests for test/UmaMerkleChefAV/Test_getMaxRemovableAmounts_UMC.t.sol:Test_getMaxRemovableIncentiveAmounts
[PASS] test_getMaxRemovableIncentiveAmounts_ActiveCampaign(uint8,uint32,uint8) (runs: 256, : 1221982, ~: 950926)
[PASS] test_getMaxRemovableIncentiveAmounts_AfterPartialRemoval(uint8,uint32,uint8,uint8) (runs: 256, : 1319588, ~:
↪  1259962)
[PASS] test_getMaxRemovableIncentiveAmounts_NearEndOfCampaign(uint8,uint32,uint8) (runs: 256, : 1300921, ~: 953449)
[PASS] test_getMaxRemovableIncentiveAmounts_PartialIncentives(uint8,uint32,uint8) (runs: 256, : 1611007, ~: 1518434)
[PASS] test_getMaxRemovableIncentiveAmounts_UnstartedCampaign(uint8,uint32,uint32) (runs: 256, : 1321895, ~: 1231974)
Suite result: ok. 5 passed; 0 failed; 0 skipped; finished in 4.89s (14.36s CPU time)

Ran 3 tests for test/UmaMerkleChefAV/Test_RemovingIncentives_UMC.t.sol:Test_RemovingIncentives_UMC
[PASS] test_RemoveIncentives_UmaMerkleChefAV(uint8,uint32,bytes) (runs: 256, : 2793805, ~: 2794772)
[PASS] test_RevertIf_RemoveIncentivesAfterCampaignEnded_UmaMerkleChefAV(uint8,uint32,bytes) (runs: 256, : 2711710, ~:
↪  2710809)
[PASS] test_RevertIf_RemoveIncentivesGtMax_UmaMerkleChefAV(uint8,uint32,bytes) (runs: 256, : 2644373, ~: 2640187)
Suite result: ok. 3 passed; 0 failed; 0 skipped; finished in 5.25s (10.32s CPU time)

Ran 2 tests for test/UmaMerkleChefAV/Test_AddingIncentives_UMC.t.sol:Test_AddingIncentives_UMC
[PASS] test_AddIncentives_UmaMerkleChefAV(uint8,uint32,bytes) (runs: 256, : 3391073, ~: 3338832)
[PASS] test_RevertIf_AddIncentivesAfterCampaignEnded_UmaMerkleChefAV(uint8,uint32,bytes) (runs: 256, : 3108684, ~:
↪  3054233)
Suite result: ok. 2 passed; 0 failed; 0 skipped; finished in 5.30s (7.05s CPU time)

Ran 1 test for test/UmaMerkleChefAV/Test_CampaignCreation_UMC.t.sol:Test_CampaignCreation_UMC
[PASS] test_IncentiveCampaignCreation_UmaMerkleChefAV(address,uint32,uint32,uint8) (runs: 256, : 1373225, ~: 1272402)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 160.77s (159.03s CPU time)
proptest: Saving this and future failures in cache/fuzz/failures
proptest: If this test was run on a CI system, you may wish to add the following line to your copy of the file. (You may
↪  need to create it.)
cc 2148bd06a919df3e55e7632715c5a069422992efafb97e152d27da82cffda5bb

Ran 3 tests for test/IncentiveLocker/Test_IncentiveLocker.sol:Test_IncentiveLocker
[PASS] test_IncentiveCampaignCreation(address,uint8,bytes) (runs: 256, : 1183847, ~: 997741)
[PASS] test_IncentiveLockerDeployment(address,address,uint64) (runs: 256, : 3041896, ~: 3041896)
[PASS] test_RevertIf_ZeroIncentiveAmountOffered(uint8) (runs: 256, : 846811, ~: 708660)
Suite result: ok. 3 passed; 0 failed; 0 skipped; finished in 210.39s (147.08s CPU time)

proptest: Saving this and future failures in cache/fuzz/failures
proptest: If this test was run on a CI system, you may wish to add the following line to your copy of the file.
cc 4774a8c6d737d463b99aa95ce18cd89b4aae4b24d608a7ac17128580934f05be
```

```
Ran 4 tests for test/UmaMerkleChefAV/Test_UmaMerkleOracleBase_UMC.t.sol:Test_UmaMerkleOracleBase_UMC
[FAIL: panic: arithmetic underflow or overflow (0x11); counterexample:
↪  calldata=0x8cd361d5000000000000000000000000000000000000000000000000000000000000000023e8000000000000000000000000000000
↪  0000000000000000000000000000026f0000000000000000000000000000000000000000000000000000000000000000600000000000000000
↪  0000000000000000000000000000000000000000000000160000000000000000000000000000000000000000000000000000000000000342a000
↪  0000000000000000000000000000000000000000000de0b6b3a763ffff0000000000000000000000000000000000000000000000000000000000
↪  00000007f5c0000000000000000000000000000000000000000000000000000000000002a280000000000000000000000000000000000000000
↪  0000000000000000000022b3000000000000000000000000000000000000000000000000000000000056c2d2eb0000000000000000000000000
↪  0000000000000000000000000000000020250000000000000000000000000000000000000000000000000000000000000043a000000000000
↪  0000000000000000000000000000000000000000000001bff0000000000000000000000000000000000000000000000000000000000000002
↪  df8000000000000000000000000000000000000000000000000000000000000169300000000000000000000000000000000000000000000000000
↪  0000000000000135000000000000000000000000000000000000000000000000000000000002bb8000000000000000000000000000000000000
↪  0000000000000000000000021de0000000000000000000000000000000000000000000000000000000000002fa3000000000000000000000000
↪  0000000000000000000000000de0b6b3a7640000000000000000000000000000000000000000000000000000000000021fe00000
↪  0000000000000000000000000000000000000000000037940000000000000000000005d932de3d050b91d0e14ac70fc4a0c
↪  c06d45af900000000000000000000000000000000000000000000000000000011e0000000000000000000000a0b86991c6218b36c
↪  1d19d4a2e9eb0ce3606eb480000000000000000000000000000000000000000000000000024cd
↪  args=[0x00000000000000000000000000000000000000000000000000000000023e8,
↪  0x00000000000000000000000000000000026f0, [0x0000000000000000000000000000342a,
↪  0x0000000000000000000000De0b6b3A763fFFf, 0x00000000000000000000000000007f5C,
↪  0x0000000000000000000000000000002A28, 0x00000000000000000000000000022B3,
↪  0x00000000000000000000000056C2d2EB, 0x0000000000000000000000000002025,
↪  0x000000000000000000000000043a0, 0x0000000000000000000000000001bfF,
↪  0x00000000000000000000000000002DF8, 0x0000000000000000000000000001693,
↪  0x0000000000000000000000000135, 0x00000000000000000000000002BB8,
↪  0x000000000000000000000000000021DE, 0x000000000000000000000000002FA3,
↪  0x0000000000000000000000de0b6B3a7640000, 0x00000000000000000000000021Fe,
↪  0x0000000000000000000000000000003794, 0x05D932dE3D050b91d0E14ac70fc4a0CC06D45af9,
↪  0x00000000000000000000000000000011e0, 0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48,
↪  0x000000000000000000000000000000024Cd]]] test_DisputeMerkleRootAssertion_UMC(bytes32,address,address[])
↪  (runs: 210, : 6078863, ~: 6121645)
[PASS] test_SuccessfullyAssertMerkleRoot_UMC(bytes32,address,address[]) (runs: 256, : 6128324, ~: 6087190)
[PASS] test_UmaMerkleOracleBaseDeployment(address,address,address[],uint64) (runs: 256, : 5407107, ~: 5542455)
[FAIL: EvmError: MemoryOOG; counterexample: calldata=0xa1e7443a000000000000000000000000000000000000000000000000000000000000000
↪  0000005920000000000000000000000000000000000000000000000000000000000006487000000000000000000000000000000000000000000
↪  0000000000000000000060000000000000000000000000000000000000000000000000000000000000043000000000000000000000000000000
↪  0000000000000000000000000000005f1500000000000000000000000000000000000000000000000000000000000017d0000000000000
↪  0000000000000000000000000000000000000000000000398d0000000000000000000000000000000000000000000000000000000000000bf1ab37
↪  400000000000000000000000000000000000000000000000000000000003e84000000000000000000000cd0de2d3fa1faef420bfe791d
↪  8c362d765e22701000000000000000000000000000000000000000000000000000000000000001f750000000000000000000000000000000000
↪  0000000000000000000003db2000000000000000000000000000000000000000000000000000000000000543a0000000000000000000000000
↪  0000000000000000000000000000000006bd70000000000000000000000000000000000000000000000000000000000004c0c0000000000000
↪  0000000000000000000000000000000000000000000013bc0000000000000000000000000000000000000000000000000000000000000000000
↪  0000b3600000000000000000000000000000000000000000000000000000000037550000000000000000000000000000000000000000000000000
↪  00000000000000001bdb0000000000000000000000000000000000000000000000000000000008da5cb5b0000000000000000000000000000000
↪  0000000000000000000000000004e0500000000000000000000000000000000000000000000000000000000000000026900000000000000000
↪  0000000000000000000000000000000000000000000022e0000000000000000000000000000000000000000000000000000000000008e7c3e70
↪  0000000000000000000000000000000000000000000000000000000e3f0000000000000000000000000000000000000000000000000000000000
↪  0000000009340000000000000000000000000000000000000000000000000000000000000c70000000000000000000000000000000000000000
↪  00000000000000000000042dd0000000000000000000000000000000000000000000000000000000000006c1000000000000000000000000000
↪  0000000000000000000000000000000006fb4000000000000000000000000000000000000000000000000000000000000001dfd000000000000
↪  0000000000000000000000000000000000000000000036f70000000000000000000000000000000000000000000000000000000000000000000
↪  0526700000000000000000000000000000000000000000000000004ac0000000000000000000000000000000000000000000000000000000000
↪  00000000000000006c1000000000000000000000000000000000000000000000000000000000000f0900000000000000000000000000000000
↪  0000000000000000000000000000010000000000000000000000000000000000000000000000000000000000051b0000000000000000000000
↪  0000000000000000000000000000000000000444000000000000000000000000000000000000000000000000000000000000002f490000000000
↪  0000000000000000000000000000000000000000000000000000000318800000000000000000000000000000000000000000000000000000000000
↪  000000022b2000000000000000000000000000000000000000000000000000000000000012ce0000000000000000000000000000000000000000
↪  0000000000000000000026f60000000000000000000000000000000000000000000000000000000000000000304300000000000000000000000004
↪  d5f4641494c4544000000000000000000000000100000000000000000000000000000000000000000000000000000022a100000000000
↪  0000000000000000000000000000000000000000000000000e990000000000000000000007f72d758e1cf3343756d543badbf24212bed8
↪  c1400000000000000000000000000000000000000000000000000000000000002d3000000000000000000000000000000000000000000000000
↪  0000000000000302d0000000000000000000000000000000000000000000000000000000003f150000000000000000000000000000000000000
↪  0000000000000000000000000091a0000000000000000000000000000000000000000000000000000000000044030000000000000000000000
↪  0000000000000000000000000000000000000001e880000000000000000000004c4544000000000000000000000000000000000000000000000
↪  0000000000000000000000000000000000000000000000000a5d0000000000000000000000000000000000000000000000000000000000000000
↪  0000034d5000000000000000000000000000000000000000000000000000000000d4eb7612000000000000000000000000000000000000000000
↪  0000000000000000003c4c0000000000000000000000000000000000000000000000000000000000224aaceb0000000000000000000000000000
↪  0000000000000000000000000000000012890000000000000000000000000000000000000000000000000000000000000051b0000000000000
↪  0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000040a
↪  700000000000000000000000000000000000000000000000000000000001ff8000000000000000000000000000000000000000000000000000000
↪  0000000000194200000000000000000000000000000000000000000000000000000000000000b2c0000000000000000000000000000000000000
↪  0000000000000000000000051a8000000000000000000000000000000000000000000000000000000000000469a0000000000000000000000000
↪  0000000000000000000000000000000000a6a22b42
```

```
args=[0x0000000000000000000000000000000000000000000000000000000000000592, 0x0000000000000000000000000000000000006487,
↪  [0x00000000000000000000000000000000005F15, 0x000000000000000000000000000000000000017D,
↪  0x000000000000000000000000000000000000398d, 0x00000000000000000000000000000000Bf1ab374, 0x0000000000000000000000000000000000003e84,
↪  0xCd0de2d3Fa1FAeF420bfe791d8C362D765E22701, 0x0000000000000000000000000000000000001F75, 0x00000000000000000000000000000000000003dB2,
↪  0x000000000000000000000000000000000000543A, 0x00000000000000000000000000000000006bD7, 0x0000000000000000000000000000000000004C0c,
↪  0x00000000000000000000000000000000000013BC, 0x00000000000000000000000000000000b36, 0x0000000000000000000000000000000000003755,
↪  0x000000000000000000000000000000000001BDB, 0x00000000000000000000000000000008Da5CB5b, 0x0000000000000000000000000000000000004e05,
↪  0x0000000000000000000000000000000000000269, 0x000000000000000000000000000000022e0, 0x00000000000000000000000000000008e7c3e7,
↪  0x00000000000000000000000000000000000e3F, 0x0000000000000000000000000000000000934, 0x0000000000000000000000000000000000C70,
↪  0x000000000000000000000000000000000042dd, 0x00000000000000000000000000000006C1, 0x0000000000000000000000000000000006fb4,
↪  0x00000000000000000000000000000000001Dfd, 0x000000000000000000000000000000036F7, 0x0000000000000000000000000000000005267,
↪  0x00000000000000000000000000000000004aC, 0x00000000000000000000000000000006C1, 0x0000000000000000000000000000000000f09,
↪  0x0000000000000000000000000000000000000001, 0x000000000000000000000000000000051b, 0x0000000000000000000000000000000004440,
↪  0x00000000000000000000000000000002F49, 0x0000000000000000000000000000003188, 0x000000000000000000000000000000000022b2,
↪  0x00000000000000000000000000000000012ce, 0x000000000000000000000000000000026F6, 0x0000000000000000000000000000000003043,
↪  0x4D5f4641494C4544000000000000000000000001, 0x000000000000000000000000000000022a1, 0x0000000000000000000000000000000000e99,
↪  0x7f72d758e1Cf3343756D543bADBf24212bed8C14, 0x00000000000000000000000000000002D30, 0x000000000000000000000000000000000302D,
↪  0x00000000000000000000000000000000003F15, 0x000000000000000000000000000000091a, 0x0000000000000000000000000000000004403,
↪  0x0000000000000000000000000000000000001E88, 0x4C4544000000000000000000000000000000, 0x00000000000000000000000000000000000A5D,
↪  0x00000000000000000000000000000000000034d5, 0x000000000000000000000000d4eb7612, 0x0000000000000000000000000000000003c4C,
↪  0x0000000000000000000000000000000224Aaceb, 0x000000000000000000000000000000001289, 0x000000000000000000000000000000000051b,
↪  0x0000000000000000000000000000000000000000, 0x00000000000000000000000000000040A7, 0x0000000000000000000000000000000001fF8,
↪  0x0000000000000000000000000000000001942, 0x00000000000000000000000000000B2C, 0x00000000000000000000000000000000051A8,
↪  0x000000000000000000000000000000000000469a, 0x000000000000000000000000A6A22B42]]]
↪  test_UnsuccessfullyAssertMerkleRoot_UMC(bytes32,address,address[]) (runs: 81, : 6046664, ~: 6075775)
Suite result: FAILED. 2 passed; 2 failed; 0 skipped; finished in 718.18s (1534.45s CPU time)

Ran 9 test suites in 718.19s (1116.23s CPU time): 30 tests passed, 2 failed, 0 skipped (32 total tests)
```

8.3 Automated Tools

8.3.1 AuditAgent

All the relevant issues raised by the AuditAgent have been incorporated into this report. The AuditAgent is an AI-powered smart contract auditing tool that analyses code, detects vulnerabilities, and provides actionable fixes. It accelerates the security analysis process, complementing human expertise with advanced AI models to deliver efficient and comprehensive smart contract audits. Available at https://app.auditagent.nethermind.io.

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 is dedicated to continuous internal research while fostering close collaboration with external partners. The team has expertise across a wide range of domains, including cryptography protocols, consensus design, decentralized identity, verifiable credentials, Sybil resistance, oracles, and credentials, distributed validator technology (DVT), and Zero-knowledge proofs. This diverse skill set, combined with strong collaboration between our engineering teams, enables us to deliver cutting-edge solutions to our 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.

Learn more about us at nethermind.io.

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.