Security Review Report
NM-0558 Royco

NETHERMIND
SECURITY

(June 11, 2025)

Contents

1 Executive Summary

This document presents the results of a security review conducted by Nethermind Security for Royco V2. The review focused on a new Action Verifier (AV) introduced to Royco using the Incentra campaigns and minor changes in `IncentiveLocker` contract.

The audit comprises 598 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 one is classified as `Medium`, and two are classified as `Info` severity. 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.

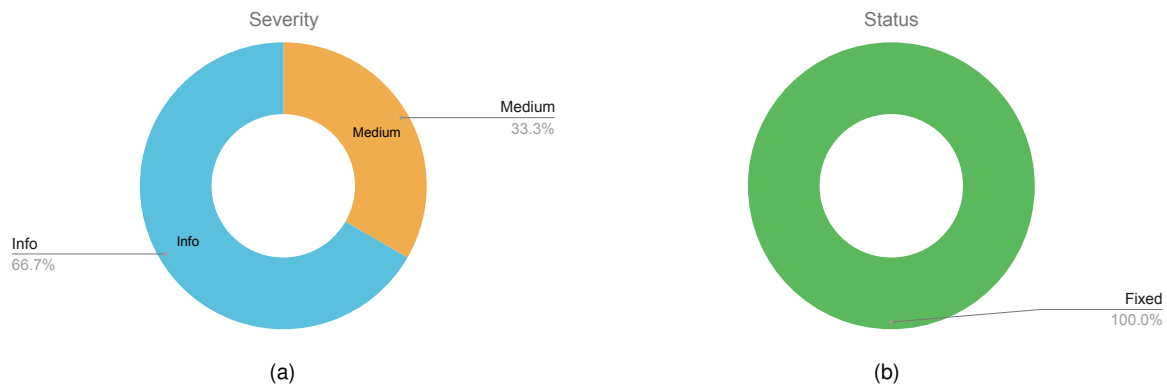

Fig. 1: Distribution of issues: Critical (0), **High** (0), **Medium** (1), **Low** (0), **Undetermined** (0), **Informational** (2), **Best Practices** (0). **Distribution of status: Fixed** (3), **Acknowledged** (0), **Mitigated** (0), **Unresolved** (0)

Summary of the Audit

Audit Type	Security Review
Initial Report	June 10, 2025
Final Report	June 11, 2025
Repository	royco-v2
Initial Commit	69d6fc08f30dd8e0155273cbc13334c17ed6416e
Final Commit	88260f360e7065b97206caca063499ef306428e0
Documentation	Royco Incentra Docs and README
Documentation Assessment	High
Test Suite Assessment	Medium

2 Audited Files

	Contract	LoC	Comments	Ratio	Blank	Total
1	src/core/IncentiveLocker.sol	399	257	64.4%	93	749
2	src/core/base/PointsRegistry.sol	82	93	113.4%	27	202
3	src/core/action-verifiers/base/ActionVerifierBase.sol	13	11	84.6%	5	29
4	src/core/action-verifiers/brevis/IncentraAV.sol	104	50	48.1%	18	172
	Total	**598**	**411**	**68.7%**	**143**	**1152**

3 Summary of Issues

	Finding	Severity	Update
1	Incentra incentives with `CROSS_CHAIN` rewards campaign can never be created	Medium	Fixed
2	Malicious Incentive Provider can block reward claims by transferring Points Program ownership to the zero address	Info	Fixed
3	Passing a non-existent incentive with zero amount incorrectly removes the last incentive	Info	Fixed

4 System Overview

For a comprehensive understanding of the entire Royco V2 system, please refer to the official audit report: NM0485-FINAL_ROYCO_V2. This section focuses exclusively on the newly integrated Action Verifier (AV) utilizing Incentra Campaigns and the corresponding minor adjustments to the `IncentiveLocker` contract.

4.1 Incentra Action Verifier (AV) Integration

When a Royco V2 campaign is created that leverages the Incentra Action Verifier, the `IncentiveLocker` contract initiates the campaign creation process by calling `processIncentiveCampaignCreation(...)` within the `IncentraAV`. A critical aspect of this integration is that when an Incentive Provider (IP) passes the corresponding Incentra Campaign address during campaign creation in Royco V2, the `IncentraAV` includes a check to ensure that the `externalPayoutAddress` in the associated Incentra Campaign is indeed set to the `IncentraAV` itself. This design ensures that the `IncentraAV` directly manages reward distribution, rather than the Incentra Campaign.

It's important to note that incentives cannot be added to a campaign **after** its initial creation when utilizing the `IncentraAV`.

4.2 Incentra Campaign Types

The `IncentraAV` supports two primary types of Incentra Campaigns, accommodating various reward proof submission mechanisms:

- **Same Chain**: In this configuration, reward proofs are submitted and processed on the same blockchain. Consequently, claiming rewards does not require the submission of Merkle proofs.
- **Cross Chain**: For scenarios where reward proofs are submitted on a different blockchain (typically a more cost-efficient chain), claiming rewards necessitates the provision of Merkle proofs.

4.3 Incentive Removal and Claiming

Incentra Campaigns provide a mechanism for Incentive Providers (IPs) to remove unclaimed incentives after a campaign concludes. This removal typically becomes available after a default period of 180 days. Therefore, it is crucial for Action Providers (APs) to claim their deserved rewards **before** this incentive removal window expires.

4.4 IncentiveLocker Enhancements

The `IncentiveLocker` contract has been updated with two key enhancements:

- **Maximum Incentive Cap**: A new restriction has been implemented, limiting the maximum amount of incentives that a particular campaign can hold.
- **Co-IP Incentive Removal**: The `IncentiveLocker` now supports Co-Incentive Providers (Co-IPs) in the process of removing incentives. However, it is the responsibility of the integrated Action Verifiers to revert the transaction if they do not permit such an action by a Co-IP.

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 [Medium] Incentra incentives with CROSS_CHAIN rewards campaign can never be created

File(s): src/core/action-verifiers/brevis/IncentraAV.sol

Description: Incentra incentives support two types of campaigns: SAME_CHAIN and CROSS_CHAIN, depending on where the rewards are submitted. When a new incentive campaign is created, the processIncentiveCampaignCreation hook is invoked. This hook verifies that the externalPayoutAddress in the Incentra campaign is set to the IncentraAV contract address:

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

        // ...
        // Check that the AV can process claims and refunds for this campaign
        require(IIncentraCampaign(params.incentraCampaign).externalPayoutAddress() == address(this),
        ↪  IncentraPayoutAddressMustBeAV());

        // ...
    }
```

However, only campaigns of type SAME_CHAIN in the Brevis contracts expose the public externalPayoutAddress function. For CROSS_CHAIN campaigns, the external payout address is instead encoded in side the config struct and not accessible via the same interface. This misalignment causes the processIncentiveCampaignCreation function to revert for every CROSS_CHAIN campaign, effectively making it impossible to create such campaigns.

Recommendation(s): Consider handling the retrieval of externalPayoutAddress differently for campaigns of type CROSS_CHAIN.

Status: Fixed

Update from the client: Fixed on Incentra contracts: 055c33c

6.2 [Info] Malicious Incentive Provider can block reward claims by transferring Points Program ownership to the zero address

File(s): src/core/base/PointsRegistry.sol

Description: The `claimIncentives(...)` function in the `IncentiveLocker` contract allows an Action Provider (AP) to claim rewards from an incentive campaign. These rewards can be standard ERC20 tokens or non-transferable points from a Points Program. The contract iterates through all owed incentives and distributes them.

The `_pushIncentivesAndAccountFees(...)` function uses the `isPointsProgram(incentive)` check to distinguish between an ERC20 token and a Points Program. If this returns `true`, it calls the internal `_award(...)` function. If it returns `false`, it assumes the incentive is an ERC20 token and performs a `safeTransfer(...)`.

```
function _pushIncentivesAndAccountFees(
    address incentive,
    address ap,
    // ...
) internal {
    // @audit Checks if the incentive is a Points Program.
    if (isPointsProgram(incentive)) {
        // ...
        _award(incentive, ap, incentiveAmount);
    } else {
        // ...
        // @audit If isPointsProgram returns false, it assumes `incentive` is an ERC20.
        ERC20(incentive).safeTransfer(ap, incentiveAmount);
    }
}
```

The `isPointsProgram(...)` function, inherited from `PointsRegistry`, determines a program's existence by checking if it has a non-zero owner address.

```
function isPointsProgram(address _pointsId) public view returns (bool exists) {
    // @audit-issue Existence is tied to the owner not being the zero address.
    exists = pointsIdToProgram[_pointsId].owner != address(0);
}
```

The vulnerability arises because the `transferPointsProgramOwnership(...)` function in the `PointsRegistry` contract lacks a check to prevent transferring ownership to the zero address.

```
function transferPointsProgramOwnership(address _pointsId, address _newOwner) external {
    PointsProgram storage pointsProgram = pointsIdToProgram[_pointsId];
    require(pointsProgram.owner == msg.sender, OnlyPointsProgramOwner());

    // @audit-issue The `_newOwner` can be `address(0)`, effectively disabling the program.
    pointsProgram.owner = _newOwner;

    emit PointsProgramOwnershipTransferred(_pointsId, _newOwner);
}
```

A malicious Incentive Provider (IP) can exploit this to mount a denial-of-service attack. An IP can create a campaign offering a legitimate ERC20 token and a Points Program they own as incentives. After an AP becomes eligible for rewards, but before they claim them, the IP can call `transferPointsProgramOwnership(...)` and set the new owner of their Points Program to `address(0)`.

The transaction will revert when the AP attempts to claim their rewards via `claimIncentives(...)`. The `isPointsProgram(...)` check will return `false` for the Points Program incentive, causing the contract to incorrectly treat its address as an ERC20 token and call `safeTransfer(...)` on it. This call will fail, reverting the entire transaction and preventing the AP from claiming any of their rewards, including the legitimate ERC20 tokens.

Recommendation(s): Consider adding a validation check in the `transferPointsProgramOwnership(...)` function to prevent transferring ownership of a Points Program to the zero address. Alternatively, the `isPointsProgram(...)` check could be modified to rely on an existing program property, which can't be a zero value or a dedicated `isCreated` flag, rather than the mutable `owner` address.

Status: Fixed

Update from the client: Fixed in 1b636e3

6.3 [Info] Passing a non-existent incentive with zero amount incorrectly removes the last incentive

File(s): src/core/IncentiveLocker.sol

Description: The `removeIncentives(...)` function allows an Incentive Provider (IP) or a Co-Incentive Provider (Co-IP) to remove incentives from an existing campaign. The function iterates through a provided list of incentives and their corresponding amounts to remove.

The issue's core lies in the fact that the function does not validate whether the incentive being removed is part of the campaign. If an IP provides a non-existent incentive address along with a `_incentiveAmountsToRemove` of `0`, the logic proceeds as if it's a valid operation. Specifically, the `ics.incentiveToAmountOffered[incentive]` for a non-existent incentive is `0`, and subtracting the provided `0` amount results in `0`. This incorrectly triggers the condition `ics.incentiveToAmountOffered[incentive] == 0`, leading to a call to the internal `_removeIncentiveFromCampaign(...)` function.

```
function removeIncentives(
    bytes32 _incentiveCampaignId,
    address[] memory _incentivesToRemove,
    uint256[] memory _incentiveAmountsToRemove,
    bytes memory _removalParams,
    address _recipient
) public nonReentrant {
    // ...
    for (uint256 i = 0; i < numIncentives; ++i) {
        address incentive = _incentivesToRemove[i];
        // ...
        uint256 incentiveAmountRemoved = _incentiveAmountsToRemove[i];

        if (incentiveAmountRemoved >= ics.incentiveToAmountRemaining[incentive]) {
            // ...
        } else {
            // ...
        }

        ics.incentiveToAmountOffered[incentive] -= incentiveAmountRemoved;
        // @audit If a non-existent incentive and a zero amount are passed, ics.incentiveToAmountOffered[incentive] will
        ↪  be 0.
        if (ics.incentiveToAmountOffered[incentive] == 0) {
            // @audit-issue This function is called with a non-existent incentive.
            _removeIncentiveFromCampaign(ics, incentive);
        }
        // ...
    }
    // ...
}
```

The `_removeIncentiveFromCampaign(...)` function is then called with this non-existent incentive. Inside this function, it attempts to find the provided incentive in the `_ics.incentivesOffered` array. Since the incentive does not exist, the loop to find its index will complete without finding a match. The function then unconditionally calls `_ics.incentivesOffered.pop()`, which removes the *last* incentive from the array, regardless of what it is.

```
function _removeIncentiveFromCampaign(ICS storage _ics, address _incentive) internal {
    uint256 lastIndex = _ics.incentivesOffered.length - 1;
    uint256 index = 0;
    // @audit This loop will not find the non-existent _incentive.
    for (index; index < lastIndex; ++index) {
        if (_ics.incentivesOffered[index] == _incentive) break;
    }

    if (index != lastIndex) {
        // ...
    }
    // @audit-issue The last element is popped, even if _incentive was not found.
    _ics.incentivesOffered.pop();
}
```

Recommendation(s): Consider ensuring that an incentive exists within the campaign before attempting to process its removal.

Status: Fixed

Update from the client: Fixed in d91bcf9

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 Royco Incentra Docs and README 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...
[] Compiling 95 files with Solc 0.8.28
[] Solc 0.8.28 finished in 104.61s
Compiler run successful with warnings:
Warning (2072): Unused local variable.
  --> script/CreateIncentiveCampaign.s.sol:26:9:
   |
26 |         bytes memory actionParams = abi.encode(startTimestamp, endTimestamp, ipfsCID);
   |         ^^^^^^^^^^^^^^^^^^^^^^^^^

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 --match-path IncentraAV
[] Compiling...
No files changed, compilation skipped

Ran 1 test for test/IncentraAV/Test_AddingIncentives_IncentraAV.t.sol:Test_AddingIncentives_IncentraAV
[PASS] test_RevertIf_AddingIncentives_IncentraAV(address,uint32,uint32,uint8,address) (runs: 256, : 6185323, ~: 5392188)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 471.61ms (469.24ms CPU time)

Ran 2 tests for test/IncentraAV/Test_RemovingIncentives_IncentraAV.t.sol:Test_RemovingIncentives_IncentraAV
[PASS] test_RemovingIncentives_IncentraAV(address,uint32,uint32,uint8,address) (runs: 256, : 3182993, ~: 2745979)
[PASS] test_RevertIf_RemovingIncentives_BeforeGracePeriodElapses_IncentraAV(address,uint32,uint32,uint8,address) (runs:
↪  256, : 5586158, ~: 4757925)
Suite result: ok. 2 passed; 0 failed; 0 skipped; finished in 472.38ms (816.30ms CPU time)

Ran 5 tests for test/IncentraAV/Test_CampaignCreation_IncentraAV.t.sol:Test_CampaignCreation_IncentraAV
[PASS] test_IncentiveCampaignCreation_IncentraAV(address,uint32,uint32,uint8,address) (runs: 256, : 3575056, ~: 3156067)
[PASS] test_RevertIf_IncentiveCampaignCreation_MoreThanOnce_IncentraAV(address,uint32,uint32,uint8,address) (runs: 256,
↪  : 6313683, ~: 5530411)
[PASS] test_RevertIf_IncentiveCampaignCreation_WithDiffIncentives_IncentraAV(address,uint32,uint32,uint8,address)
↪  (runs: 256, : 5772973, ~: 5040255)
[PASS]
↪  test_RevertIf_IncentiveCampaignCreation_WithDiffNumIncentives_IncentraAV(address,uint32,uint32,uint8,uint8,address)
↪  (runs: 256, : 5837516, ~: 5541536)
[PASS] test_RevertIf_IncentiveCampaignCreation_WithWrongEPA_IncentraAV(address,uint32,uint32,uint8,address,address)
↪  (runs: 256, : 3354385, ~: 2852800)
Suite result: ok. 5 passed; 0 failed; 0 skipped; finished in 474.11ms (1.98s CPU time)

Ran 3 test suites in 475.27ms (1.42s CPU time): 8 tests passed, 0 failed, 0 skipped (8 total tests)
```

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.