Security Review Report
NM-0468 Mellow



NETHERMIND
SECURITY

(March 17, 2025)

Contents

1 Executive Summary 2

2 Audited Files 3

3 Summary of Issues 3

4 Protocol Overview 4
- 4.1 Deposit Process . . . 4
- 4.2 Cross-Chain Interaction . . . 4
- 4.3 Withdrawal Process . . . 4

5 Risk Rating Methodology 5

6 Issues 6
- 6.1 [Low] Potential loss of user shares when claiming in `WithdrawalQueue` . . . 6
- 6.2 [Info] The `PUSH_ROLE` entity must have a `receive` function if smart contract . . . 6
- 6.3 [Best Practices] Incomplete comment documentation . . . 7
- 6.4 [Best Practices] Unnecessary `override` tag for limit functions . . . 7

7 Documentation Evaluation 8

8 AuditAgent 8

9 Additional Observations 9
- 9.1 Initialization . . . 9
- 9.2 Centralization risk . . . 9

10 Test Suite Evaluation 9
- 10.1 Compilation Output . . . 9
- 10.2 Tests Output . . . 9

11 About Nethermind 10

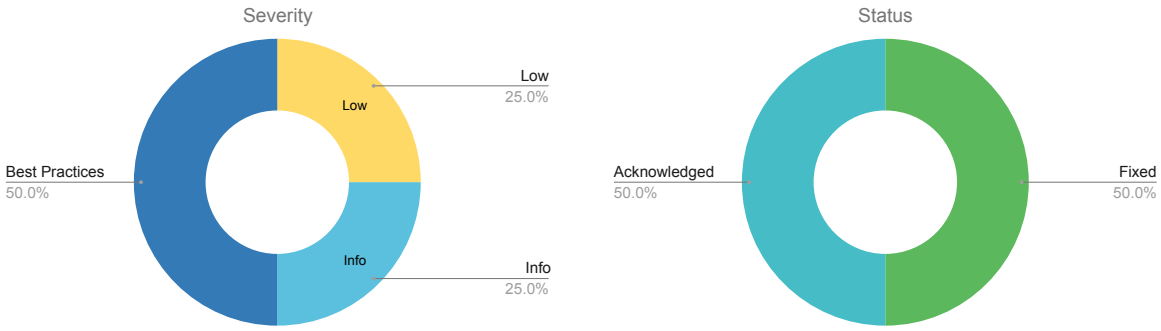
1 Executive Summary

This document outlines the security review performed by Nethermind Security for the Mellow Interop smart contracts. These contracts are designed to allow users to delegate stake without direct liquidity transfers across various L2s to L1 (Ethereum). To achieve liquidity transfers between chains the LayerZero OFT is used, with a small modification on the L2 adapter contract to restrict deposits to trusted contracts.

The audited code comprises of 709 lines of code written in the Solidity language, and the audit was performed using (a) manual analysis of the codebase, (b) automated analysis tools, (c) simulation of the smart contract.

Along this document, we report four points of attention, where one is classified as `Low`, and three are classified as `Informational` or `Best Practices`. 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 describes 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 details automated tooling used during the audit. Section 9 presents additional observations. Section 10 presents the compilation, tests, and automated tests. Section 11 concludes the document.



(a) distribution of issues according to the severity

(b) distribution of issues according to the status

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

Summary of the Audit

Audit Type	Security Review
Draft Report	March 17, 2025
Final Report	March 19, 2025
Methods	Manual Review, Automated analysis
Repository	mellow-finance/mellow-interop
Initial Commit Hash	150bf68ae3f03882eddc24fa67f0a986fb976851
Final Commit Hash	d7564475edcf4c6b65b5385c15725a186f130596
Documentation Assessment	Medium
Test Suite Assessment	Medium

2 Audited Files

	Contract	LoC	Comments	Ratio	Blank	Total
1	src/core/TargetCore.sol	60	6	10.0%	10	76
2	src/core/SourceCore.sol	103	10	9.7%	15	128
3	src/core/TargetCoreStorage.sol	60	11	18.3%	19	90
4	src/core/SourceCoreStorage.sol	78	12	15.4%	23	113
5	src/utils/Delegator.sol	31	5	16.1%	8	44
6	src/utils/WithdrawalQueue.sol	91	17	18.7%	18	126
7	src/utils/Oracle.sol	35	10	28.6%	9	54
8	src/oft/MellowOFT.sol	11	2	18.2%	3	16
9	src/oft/MellowOFTAdapter.sol	32	5	15.6%	7	44
10	src/interfaces/core/ISourceCoreStorage.sol	51	1	2.0%	15	67
11	src/interfaces/core/ITargetCoreStorage.sol	37	1	2.7%	14	52
12	src/interfaces/core/ITargetCore.sol	20	1	5.0%	11	32
13	src/interfaces/core/ISourceCore.sol	17	1	5.9%	8	26
14	src/interfaces/utils/IWithdrawalQueue.sol	27	1	3.7%	22	50
15	src/interfaces/utils/IDelegator.sol	18	1	5.6%	7	26
16	src/interfaces/utils/IOracle.sol	15	1	6.7%	13	29
17	src/interfaces/oft/IMellowOFT.sol	8	1	12.5%	2	11
18	src/interfaces/oft/IMellowOFTAdapter.sol	15	1	6.7%	5	21
	Total	**709**	**87**	**12.3%**	**209**	**1005**

3 Summary of Issues

	Finding	Severity	Update
1	Potential loss of user shares when claiming in `WithdrawalQueue`	Low	Acknowledged
2	The `PUSH_ROLE` entity must have a `receive` function if smart contract	Info	Acknowledged
3	Incomplete comment documentation	Best Practices	Fixed
4	Unnecessary `override` tag for limit functions	Best Practices	Fixed

4 Protocol Overview

The Mellow Interop project is designed to allow users to delegate stake without direct liquidity transfers from various Layer 2 networks, known as the Source chain, to the Layer 1 network, referred to as the Target chain. Stakes from the Source are transferred to Mellow's MultiVault in the Target chain for various yield generation strategies. The cross-chain interaction between the Source chain and Target chain is facilitated by LayerZero's OFT with slight modifications in the withdrawal process. The Mellow Interop protocol can be divided into three distinct processes:

4.1 Deposit Process

The deposit process starts at the Source chain through SourceCore.sol contract, a customized ERC4626 vault. ERC4626 is a tokenized vault standard that uses ERC20 tokens to represent ownership shares of another asset. As a result, users deposit one ERC20 token into the ERC4626 vault contract and receive another ERC20 token representing its shares in the vault. As with any ERC4626 vault, the SourceCore.sol contract allows users to either deposit assets to receive pro-rata shares through the deposit() function:

```
function deposit(uint256 assets, address receiver) external returns (uint256);
```

Additionally, users can decide to mint specific ERC4626 shares from the vault via the mint() function, which will calculate and transfer the required assets to the vault:

```
function mint(uint256 shares, address receiver) external returns (uint256);
```

4.2 Cross-Chain Interaction

To delegate stake to the Target chain as well as transfer stake back to the Source chain for withdrawal, the Mellow Interop protocol uses LayerZero. Once there are deposits in the ERC4626 SourceCore contract, the PUSH_ROLE entity can transfer assets from the Source chain to the Target chain by calling the pushToTarget() function:

```
function pushToTarget() external payable returns (uint256 assets);
```

The pushToTarget() function determines the amount of funds to transfer to the Target chain by taking the difference between the vault's total asset balance and pending withdrawals. This difference amount is pushed to Layer 1 for yield generation.

The Target chain, through the TargetCore contract, implements a mechanism to transfer tokens back to the Source chain via the pushToSource() function:

```
function pushToSource(uint256 assets) external payable;
```

TargetCore.sol is also a tokenized vault. In the event there is insufficient liquidity in the Source chain to fulfill withdrawal requests, tokens will be redeemed from the TargetCore contract and liquidity pushed back to the Source chain via the pushToSource() function.

4.3 Withdrawal Process

As mentioned, the SourceCore contract is an ERC4626 vault with a customized withdrawal process. In the SourceCore contract, withdrawing is a two-step process handled in epochs. The withdrawal process starts with users requesting withdrawals, waiting for a withdrawal delay period to pass, and finally withdrawing their assets. The Mellow Interop protocol implements the WithdrawalQueue contract, which is a utility contract to handle withdrawals from the SourceCore contract.

As a result, users would request withdrawal via the requestWithdrawal(...) function in the SourceCore contract:

```
function requestWithdrawal(uint256 shares) external;
```

The requestWithdrawal(...) function calls WithdrawalQueue's request(...) function and subsequently, shares to be withdrawn will be transferred to the WithdrawalQueue. The WithdrawalQueue contract tracks user withdrawals in shares, and on elapse of the withdrawal delay period, computes the amount of assets against the shares available for withdrawal. Users can then call the claim(...) function to withdraw their stakes:

```
function claim(uint256 epoch, address receiver) external returns (uint256 assets);
```

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] Potential loss of user shares when claiming in `WithdrawalQueue`

File(s): `WithdrawalQueue.sol`

Description: The `claim` function in the `WithdrawalQueue` contract is called by a user after they have waited for their epoch to complete, at which point they can claim their shares to receive some amount of assets. If the conversion from shares to assets results in the user getting zero assets, the function will return zero rather than reverting. This means that it is possible for a user to receive no assets and still have their shares deleted.

```
function claim(uint256 epoch, address receiver) external nonReentrant returns (uint256 assets) {
    // ...
    address account = msg.sender;
    uint256 shares_ = sharesOf[epoch][account];
    if (shares_ == 0) {
        return 0;
    }
    delete sharesOf[epoch][account];
    assets = Math.mulDiv(shares_, withdrawals[epoch], shares[epoch]);
    if (assets == 0) {
        return 0;
    }
    withdrawals[epoch] -= assets;
    shares[epoch] -= shares_;
    asset.safeTransfer(receiver, assets);
    // ...
}
```

Recommendation(s): Consider reverting when the amount of assets is zero. Alternatively, the clearing of shares using the `delete` operator can be done after the early return `if` statement.

Status: Acknowledged

Update from the client: We believe any potential loss will be negligible, likely limited to few weis, and therefore considered insignificant.

6.2 [Info] The `PUSH_ROLE` entity must have a `receive` function if smart contract

File(s): `SourceCore.sol`

Description: An entity with access to `PUSH_ROLE` will need to be able to call the function `pushToTarget` on the `SourceCore` contract. If the account with this role is a smart contract, it must have a `receive` function implemented as the refund addressed passed to the adapter is the message sender.

```
function pushToTarget() ... onlyRole(PUSH_ROLE) returns (uint256 assets) {
    //...
    (MessagingReceipt memory msgReceipt, OFTReceipt memory oftReceipt) = adapter_.send{value: msg.value}(
        SendParam(
            targetEndpointId(),
            targetCoreAddress(),
            assets,
            assets,
            new bytes(0),
            new bytes(0),
            new bytes(0)
        ),
        MessagingFee(msg.value, 0),
        _msgSender() // Refund address is the caller, must support native asset transfers
    );
    //...
}
```

If the `PUSH_ROLE` address is a smart contract that does not have a receive function, a call to `pushToTarget` that involves a refund will cause the transaction to revert.

Recommendation(s): Smart contracts with the `PUSH_ROLE` should have the ability to receive native asset payments directly.

Status: Acknowledged

Update from the client: We will initially use an EOA and later transition to a contract with a dedicated receive function.

6.3 [Best Practices] Incomplete comment documentation

File(s): `*.sol`

Description: Many files in the codebase make use of the `@inheritdoc` comment tag. However, there are multiple instances where the parent contracts do not feature any comments to be inherited. This affects readability and makes readers rely on variable naming and/or exploring logic flow to understand the protocol behavior. Consider the example shown below:

```
// FILE: `SourceCore.sol`

/// @inheritdoc ISourceCore
function initialize(InitParams calldata params) external initializer {
    __SourceCoreStorage_init(params);
}
```

```
// FILE: `ISourceCore.sol`

// @audit: There is no comment documentation for `initialize`
function initialize(InitParams calldata params) external;
```

Recommendation(s): Consider adding comments to all variables and functions which are referred to using an `@inheritdoc` tag by a child contract to improve comment documentation.

Status: Fixed

Update from the client: Fixed at commit d7564475edcf4c6b65b5385c15725a186f130596.

6.4 [Best Practices] Unnecessary `override` tag for limit functions

File(s): `SourceCoreStorage.sol`

Description: The function `limit` and `setLimit` in `SourceCoreStorage` feature the `override` tag. However, both of these functions are base implementations that do not override any existing contract logic.

```
// Unnecessary `override` tag for limit functions
function limit() public view override returns (uint256) {...}
function setLimit(uint256 limit_) external override onlyRole(...) {...}
```

Recommendation(s): Consider removing the `override` tag from the `limit` and `setLimit` functions.

Status: Fixed

Update from the client: Fixed at d7564475edcf4c6b65b5385c15725a186f130596.

7 Documentation Evaluation

Software documentation refers to the written or visual information describing software's functionality, architecture, design, and implementation. 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 Interop documentation

The `Mellow Interop` protocol provided internal documentation describing the contracts on both the `Source` chain and `Target` chain, as well as the oracle update strategy. All questions raised by the Nethermind were addressed by the Mellow team.
The inline comment documentation initially had missing descriptions for certain functions and variables, where an `@inheritdoc` tag referred to non-existing documentation. This has since been addressed, all functions and variable have descriptive comments.

8 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 Additional Observations

9.1 Initialization

Post-deployment, the contract's state is configured with initial values in the `initialize` function to make the contract ready for use. It was observed that not all necessary roles are configured during initialization, as both `SourceCore` and `TargetCore` contracts have functions that are entitled to specific roles only. Additionally, some functionalities are cross-contract and hence will not work until configured. However, since the default admin role was set, the protocol can grant any role when required.

Hence, post-deployment, all cross-contract functions may not work and would need additional configuration to set up roles for some functions to work.

9.2 Centralization risk

Due to cross-chain staking, total assets need to be tracked on SourceChain, TargetChain, and those in transit between the two chains. In order to track total assets, the protocol implements an oracle that computes the total assets based on a configurable value. A dedicated role is granted permissions to set this value, allowing it to be configured to inflate or deflate the value of assets.

This presents a potential centralization risk, as the protocol can configure the value to increase or decrease assets in its favor.

10 Test Suite Evaluation

10.1 Compilation Output

```
user@machine mellow-interop % forge build
Compiling...
Compiling 178 files with Solc 0.8.25
Solc 0.8.25 finished in 6.22s
```

10.2 Tests Output

```
kalzak@mbp mellow-interop % forge test -vvv --fork-url https://eth-mainnet.public.blastapi.io --fork-block-number
↪  21900000
Compiling...
No files changed, compilation skipped

Ran 1 test for test/IntegrationTest.t.sol:IntegrationTest
[PASS] testCompleteWorkflow() (gas: 2149732)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 28.37ms (10.32ms CPU time)

Ran 2 tests for test/SourceCore.t.sol:UnitTest
[PASS] testConstructor() (gas: 4566477)
[PASS] testIntegration() (gas: 15352181)
Suite result: ok. 2 passed; 0 failed; 0 skipped; finished in 31.78ms (28.78ms CPU time)

Ran 2 test suites in 724.62ms (60.15ms CPU time): 3 tests passed, 0 failed, 0 skipped (3 total tests)
```

11 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 procehttps://www.overleaf.com/project/65c0e737f41a29601bda5c48ss, 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.