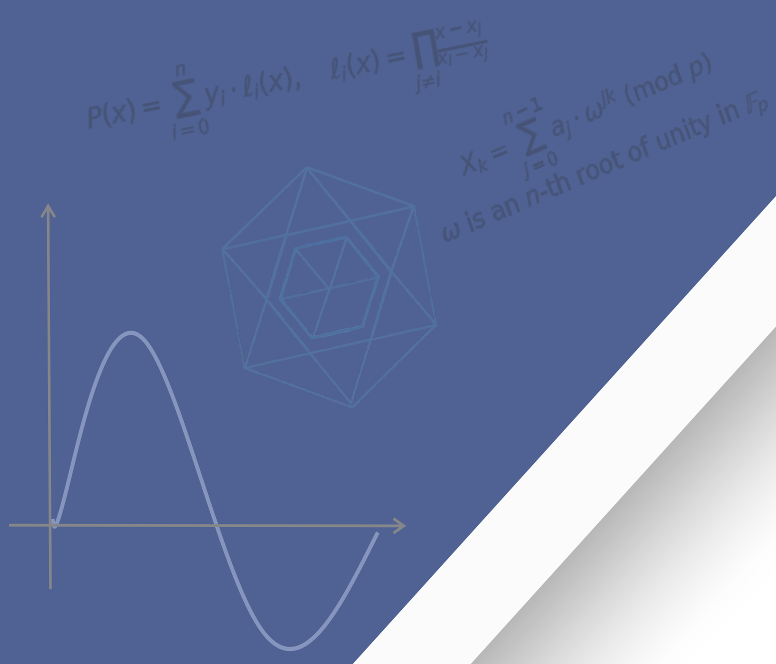

Security Review Report

NM-0679 - Celo-Hokulea
Integration

November 25, 2025

Contents

1 Executive Summary

This document presents the results of the security review conducted by Nethermind Security for Celo's integration of EigenDA Hokulea. Specifically on the `op-succint` side we review `PR 572` and a diff with the `op-succint` upstream. As well as changes made to `Celo-kona`.

This audit focused on the end-to-end review of the above described changes. **The audit comprises 1381** lines of Rust and Solidity code combined. The audit was performed using (a) manual analysis of the codebase, and (b) automated analysis tools.

Along this document, we report 4 points of attention, where 1 is classified as `Low`, and 3 are classified as `Informational` or `Best Practices` severity. The issues are summarized in Fig. 1. **It is important to note that the codebase does not include a test suite. Implementing a comprehensive suite of tests before deployment is essential to validate the protocol's behavior and ensure its correctness. We recommend that the Fileverse team develop and maintain such a suite, covering both expected workflows and edge-case scenarios.**

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.

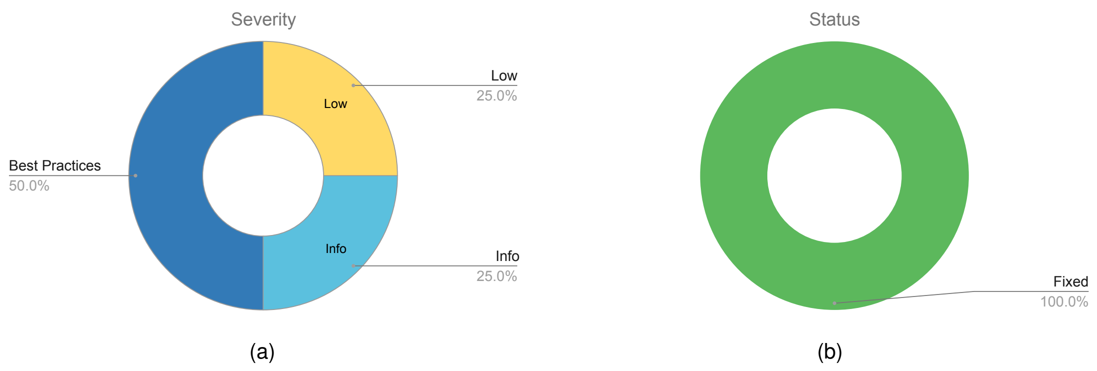

Fig. 1: Distribution of issues: Critical (0), **High** (0), **Medium** (0), **Low** (1), **Undetermined** (0), **Informational** (1), **Best Practices** (2).
Distribution of status: Fixed (4), **Acknowledged** (0), **Mitigated** (0), **Unresolved** (0)

Summary of the Audit

Audit Type	Security Review
Initial Report	October 13, 2025
Final Report	November 25, 2025
Initial `op-succint PR 572` Commit Hash	b62d805e71838a2ab1dce25876e8617e20bc8673
Initial `op-succint` diff Commit Hash	54657eb0e9ba100915808bb1c0fb72bb575eafbf
Initial `Celo-kona` Commit Hash	0a5129b4d58f4d878c96852de843fa813fe84113
Final `op-succint` Commit Hash	fb8a7766f699af5f04c381b54156906dfd240936
Final `Celo-kona` Commit Hash	a90700bd2853d8f10b7f292b5efe3a2a5b0e1320
Documentation Assessment	High
Test Suite Assessment	High

Summary of Engagement

Dates	Methods	Consultants Engaged	Level of Effort
Sept. 29 - Oct. 10, 2025	Manual	2	2-person weeks

2 Audited Files

2.1 op-succint PR 572

	Contract	LoC
1	fault-proof/src/proposer.rs	1
2	programs/range/eigenda/Cargo.toml	35
3	programs/range/eigenda/src/main.rs	44
4	scripts/prove/bin/agg.rs	8
5	scripts/prove/src/lib.rs	8
6	utils/build/src/lib.rs	5
7	scripts/utils/bin/cost_estimator.rs	2
8	utils/client/src/client.rs	5
9	utils/client/src/witness/mod.rs	20
10	utils/eigenda/client/Cargo.toml	28
11	utils/eigenda/client/src/executor.rs	76
12	utils/eigenda/client/src/lib.rs	1
13	utils/eigenda/host/Cargo.toml	37
14	utils/eigenda/host/src/host.rs	85
15	utils/eigenda/host/src/witness_generator.rs	2
16	utils/eigenda/host/src/witness_generator.rs	181
17	utils/elfs/src/lib.rs	5
18	utils/host/Cargo.toml	3
19	utils/host/src/host.rs	15
20	utils/proof/Cargo.toml	2
21	utils/proof/src/lib.rs	14
22	validity/Cargo.toml	1
23	validity/Dockerfile.eigenda	65
24	validity/src/proof_requester.rs	6
Total		**649**

2.2 op-succint diff

	Contract	LoC
1	contracts/script/fp/ConfigureDeploymentSafe.s.sol	147
2	contracts/script/fp/DeployOPSuccinctFDG.s.sol	137
3	contracts/script/fp/UpgradeOPSuccinctFDG.s.sol	10
4	contracts/test/helpers/JSONDecoder.sol	5
5	fault-proof/bin/proposer.rs	2
6	fault-proof/src/config.rs	9
7	fault-proof/src/lib.rs	12
8	fault-proof/src/proposer.rs	7
9	fault-proof/tests/common/env.rs	5
10	fault-proof/tests/common/process.rs	1
11	programs/range/utils/src/lib.rs	20
12	scripts/prove/bin/multi.rs	2
13	scripts/utils/bin/cost_estimator.rs	3
14	scripts/utils/bin/fetch_fault_dispute_game_config.rs	44
15	utils/celestia/client/src/executor.rs	11
16	utils/client/src/boot.rs	10
17	utils/client/src/client.rs	37
18	utils/client/src/witness/executor.rs	42
19	utils/eigenda/client/src/executor.rs	11
20	utils/eigenda/host/src/host.rs	8
21	utils/eigenda/host/src/witness_generator.rs	21
22	utils/ethereum/client/src/executor.rs	11
23	utils/ethereum/host/src/host.rs	10
24	utils/host/src/fetcher.rs	27
25	utils/host/src/host.rs	5
26	utils/host/src/witness_generation/traits.rs	20
Total		**617**

2.3 Celo-kona

	Contract	LoC
1	transfer.rs	115
Total		**115**

3 Summary of Issues

	Finding	Severity	Update
0	Insecure crate used	Low	Fixed
1	Duplicate code leading to redundant cycles	Info	Fixed
2	Gas behavior differs if precompile is used outside ‘GoldToken‘	Best Practice	Fixed
3	Missing STATICCALL guard allows state-changing precom pile to be invoked in static context	Best Practice	Fixed

4 System Overview

4.1 PR 572

The primary objective of PR 572 is to implement comprehensive Eigen DA support within the Hokulea component of the OP Succinct stack. Specifically, it extends Hokulea's derivation and batch-processing pipeline to natively interact with EigenDA as a data availability backend. This includes enabling the system to submit, retrieve, and verify data blobs via the EigenDA proxy, manage KZG commitment verification, and ensure robust failover handling.

By introducing these capabilities, the PR allows rollups built on OP Succinct to seamlessly operate using EigenDA as their data availability layer, thereby enhancing modularity, scalability, and interoperability across different DA solutions.

4.2 op-succint diff

The OP-Succinct zero-knowledge proof system has been updated with several key shifts. First, it now supports the Celo blockchain instead of just Optimism. All core types were replaced to integrate Celo's OP Stack-based L2, enabling validity proofs for Celo transactions that include its custom EVM extensions and transaction formats.

The ConfigureDeploymentSafe.s.sol script batches configuration calls via Safe delegate transactions, while the main script (DeployOPSuccinctFDG.s.sol) now separates the rollout into three clear stages: deploy, configure, and activate.

Finally, the SP1 prover setup moved from FulfillmentStrategy::Hosted (which skipped simulation) to FulfillmentStrategy::Reserved (with simulation enabled). This ensures more rigorous validation and guarantees reserved prover capacity. A new configurable cycle limit parameter—defaulting to 3 billion cycles—was also introduced for finer-grained control over proof execution.

4.3 Celo-kona

The transfer precompile was reviewed. The transfer precompile enables the sending of the native token of a different user to that who is calling the contract on-chain.

The precompile takes in the context, input, caller address, and the gas limit. The initial checks it performs are that the gas cost is greater than the minimum transfer gas cost of 9 thousand, that the caller address is the celo token address and that the input length is 96 bytes. The input contains the from, to and value values in one array.

Then it stores the warmth (cold or warm) of the to address and performs the transfer based on the input values. This applies a state change to the journal. Finally, it resets the warmth of the address to its initial value.

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.

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

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 development best practices;

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

6 Issues

6.1 [Low] Insecure crate used

File(s): `cargo.toml`

Description: Usage of `tracing-subscriber` package version `0.3.19` is deemed insecure as logging user input may result in poisoning logs with ANSI escape sequences explained in RUSTSEC.

Recommendation(s): Upgrade to version `>=0.3.20`.

Status: Fixed

Update from the client: Created patch PR in both celo-kona and op-succinct:
PR 78
PR 33

6.2 [INFO] Duplicate code leading to redundant cycles

File(s): `main.rs`, `lib.rs`

Description: In both `main.rs` and `lib.rs/run_range_program(...)` the function `get_oracle_and_blob_provider(...)` is called to create an oracle and a blob store. This means the same code is executed twice and can thus slow down the proving time of the code.

Recommendation(s): Editing the function `run_range_program(...)` to take as parameters the oracle and blob store, as this would mean that it would not need to be recalculated.

Status: Fixed

Update from the client: Included in the following PR: PR 38.

6.3 [Best Practice] Gas behavior differs if precompile is used outside `GoldToken`

File(s): `precompiles/transfer.rs`

Description: The transfer precompile's `run(...)` function restores the warmness of the `to` address after performing a balance transfer but does not restore the warmness of the `from` address. In the current system, this behavior is correct and has no side effects because the precompile is always invoked through the `GoldToken` contract.

Before calling the precompile, `GoldToken` performs a balance check on the `from` address, which triggers the EVM `BALANCE` opcode:

```
function balanceOf(address owner) public view returns (uint256) {
    //@audit BALANCE opcode warms `owner`
    return owner.balance;
}
```

According to EVM semantics, any address accessed via the `BALANCE` opcode is automatically added to the access list and becomes warm. Therefore, by the time the precompile executes, the `from` account is already warm, and restoring its warmness state is unnecessary.

However, this assumption only holds as long as the precompile is invoked through `GoldToken`. If the precompile were to be called directly by another contract or address that does not access `from` beforehand, the `from` address could start as cold. In such a case, the internal `journal.transfer(...)` call would mark it warm and alter gas metering for the remainder of the transaction.

Recommendation(s): Consider documenting that the `from` address is guaranteed to be warm when the precompile is called through `GoldToken`.

Status: Fixed

Update from the client: Made a PR for it and merged PR 77.

6.4 [Best Practice] Missing STATICCALL guard allows state-changing precompile to be invoked in static context

File(s): `precompiles/transfer.rs`

Description: The `transfer_run(...)` flow exposes a state-changing precompile that debits and credits balances through `journal().transfer(...)`. This precompile is intended to be called only by the token contract in standard, non-static execution, and it charges gas before returning. However, the implementation does not verify whether the call is being executed under `STATICCALL` (static context). In static context, EVM semantics prohibit state changes. Despite that, the code proceeds to execute the transfer, which attempts to mutate state.

```
fn run<CTX>(
    context: &mut CTX,
    input: &Bytes,
    caller_address: Address,
    gas_limit: u64,
) -> PrecompileResult
where
    CTX: ContextTr<Cfg: Cfg<Spec = OpSpecId>>,
{
    // ...
    if input.len() != 96 {
        return Err(PrecompileError::Other("invalid input length".to_string()));
    }
    // ...
    // @audit This call mutates state but there is no STATICCALL guard before it.
    let result = context.journal().transfer(from, to, value);
    // ...
}
```

Recommendation(s): Consider introducing a strict check that rejects execution when the precompile is invoked in a static context. The precompile should exit early with an error indicating that state changes are not allowed during static calls.

Status: Fixed

Update from the client: Opened a PR PR 79.

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. SDK's 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 SDK'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 SDK. 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 SDK. 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 SDK. 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 SDK 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 SDK. This type of documentation is critical in ensuring that the SDK is secure and free from vulnerabilities.

These types of documentation are essential for SDK 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 Celo's documentation

`Celo's` 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

```
> cargo test
test contracts::erc20::tests::test_credit_gas_fees_calldata_structure ... ok
test contracts::core_contracts::tests::test_get_intrinsic_gas ... ok
test api::default_ctx::test::default_run_celo ... ok
test contracts::core_contracts::tests::test_get_currencies ... ok
test contracts::core_contracts::tests::test_get_exchange_rates ... ok
test common::fee_currency_context::tests::test_new_from_evm ... ok
test contracts::erc20::tests::test_debit_gas_fees_calldata_structure ... ok
test evm::tests::test_deposit_tx ... ok
test evm::tests::test_halted_deposit_tx ... ok
test evm::tests::test_halted_tx_call_bls12_381_g1_add_input_wrong_size ... ok
test evm::tests::test_halted_tx_call_bls12_381_g1_msm_input_wrong_size ... ok
test evm::tests::test_halted_tx_call_bls12_381_g1_add_out_of_gas ... ok
test evm::tests::test_halted_tx_call_bls12_381_g1_msm_out_of_gas ... ok
test evm::tests::test_halted_tx_call_bls12_381_g1_msm_wrong_input_layout ... ok
test evm::tests::test_halted_tx_call_bls12_381_g2_add_input_wrong_size ... ok
test evm::tests::test_halted_tx_call_bls12_381_g2_add_out_of_gas ... ok
test evm::tests::test_halted_tx_call_bls12_381_g2_msm_out_of_gas ... ok
test evm::tests::test_halted_tx_call_bls12_381_g2_msm_input_wrong_size ... ok
test evm::tests::test_halted_tx_call_bls12_381_map_fp2_to_g2_out_of_gas ... ok
test evm::tests::test_halted_tx_call_bls12_381_map_fp_to_g1_out_of_gas ... ok
test evm::tests::test_halted_tx_call_bls12_381_map_fp2_to_g2_input_wrong_size ... ok
test evm::tests::test_halted_tx_call_bls12_381_pairing_input_wrong_size ... ok
test evm::tests::test_halted_tx_call_bls12_381_g2_msm_wrong_input_layout ... ok
test evm::tests::test_halted_tx_call_bls12_381_pairing_out_of_gas ... ok
test evm::tests::test_halted_tx_call_bls12_381_map_fp_to_g1_input_wrong_size ... ok
test evm::tests::test_tx_call_bls12_381_pairing_wrong_input_layout ... ok
test evm::tests::test_tx_call_p256verify ... ok
test handler::tests::test_consume_gas ... ok
test handler::tests::test_consume_gas_deposit_tx ... ok
test evm::tests::test_halted_tx_call_bn128_pair_granite ... ok
test handler::tests::test_commit_mint_value ... ok
test handler::tests::test_consume_gas_sys_deposit_tx ... ok
test evm::tests::test_log_inspector ... ok
test handler::tests::test_consume_gas_with_refund ... ok
test handler::tests::test_operator_fee_refund::case_1_deposit ... ok
test evm::tests::test_halted_tx_call_p256verify ... ok
test handler::tests::test_operator_fee_refund::case_2_dyn_fee ... ok
test handler::tests::test_remove_l1_cost_non_deposit ... ok
test handler::tests::test_remove_l1_cost ... ok
test evm::tests::test_halted_tx_call_bn128_pair_fjord ... ok
test handler::tests::test_remove_operator_cost ... ok
test handler::tests::test_remove_l1_cost_lack_of_funds ... ok
test handler::tests::test_validate_sys_tx ... ok
test handler::tests::test_revert_gas ... ok
test precompiles::tests::test_celo_precompiles_count ... ok
test precompiles::tests::test_transfer ... ok
test handler::tests::test_validate_deposit_tx ... ok
test precompiles::transfer::tests::test_basic_transfer_cold_account ... ok
test precompiles::transfer::tests::test_basic_transfer_oog ... ok
test precompiles::transfer::tests::test_basic_transfer_success ... ok
test handler::tests::test_validate_tx_against_state_deposit_tx ... ok
test transaction::abstraction::tests::test_cip64_transaction_fields ... ok
test precompiles::tests::test_token_duality ... ok

test result: ok. 53 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.00s
```

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