Security Review Report
NM-0691 - Spiko MultiATM

NETHERMIND
SECURITY

(November 3, 2025)

Contents

1 Executive Summary

This document presents the results of a security review conducted by Nethermind Security for Spiko MultiATM contract.

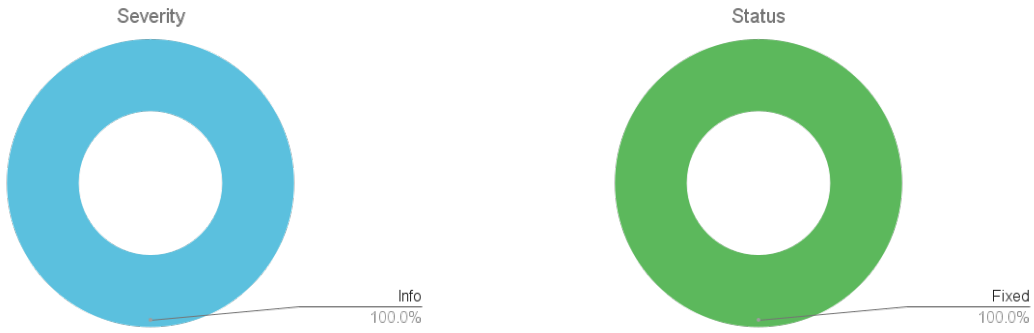
Spiko is a protocol for tokenizing securities on public blockchains, specifically focused on money market funds. The protocol enables the first fully-licensed money market funds in the EU to be issued on-chain.

MultiATM is a decentralized token swap contract designed as an Automated Token Machine (ATM) that facilitates atomic token exchanges using oracle-based pricing. As such, the contract enables multi-hop swap capabilities and supports meta-transactions.

The audit comprises 203 lines of Solidity code. **The audit was performed using** (a) manual analysis of the codebase, and (b) automated analysis tools.

Along this document, we report two points of attention, which are classified as `Informational`. The issues are summarized in Fig. 1.

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



(a) (b)

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

Summary of the Audit

Audit Type	Security Review
Initial Report	October 22, 2025
Final Report	November 3, 2025
Initial Commit	42911b8
Final Commit	f283264
Documentation Assessment	High
Test Suite Assessment	High

2 Audited Files

	Contract	LoC	Comments	Ratio	Blank	Total
1	token/MultiATM.sol	203	36	17.7%	37	276
	Total	**203**	**36**	**17.7%**	**37**	**276**

3 Summary of Issues

	Finding	Severity	Update
1	Lack of Slippage Protection can lead to Unfavorable Exchange Rates	Info	Fixed
2	The contract defaults to 18 decimals if the token does not implement the `decimals(...)` function.	Info	Fixed

4 Protocol Overview

`MultiATM` is a decentralized token swap contract designed as an Automated Token Machine (ATM) that facilitates atomic token exchanges using oracle-based pricing. The contract implements an intricate pricing mechanism with Chainlink-style oracles, multi-hop swap capabilities, and meta-transaction support.

The `MultiATM` smart contract extends `ERC2771` to support meta transactions. Additionally, the contract extends `Multicall` to allow batching of multiple function calls in a single transaction. The contract implements role-based access control.

4.1 Token Pair

The `MultiATM` contract registers token pairs and uses a `commutative hash` of both token addresses for identification, this enables bidirectional token swap using the same token configuration.

```
struct Pair {
       IERC20 token1;
       IERC20 token2;
       Oracle oracle;
       uint256 oracleTTL;
       uint256 numerator;
       uint256 denominator;
   }
```

4.2 Price Discovery

The contract uses Chainlink-compatible oracles for price discovery and implements the following functionalities.

- Uses the `_getPrices()` function to read the price information for conversion of input tokens to output tokens
- Uses two price points (latest and previous) for slippage protection
- TTL validation prevents stale price usage
- Min/max bounds provide natural slippage protection

4.3 Swap Execution

The MultiATM contract supports the following swap types

- **Exact Input Swaps:**
  - The user specifies the exact input amount
  - The contract calculates the output amount using Math.Rounding.
  - Floor for conservative output
- **Exact Output Swaps:**
  - The user specifies the exact output amount
  - Contract calculates required input Uses Math.Rounding.
  - Ceil for conservative input
- **Single vs Multi-hop:**
  - Single: Direct token-to-token swap
  - Multi-hop: Path-based swaps through multiple pairs

4.4 Fee Mechanism

Fees configuration and calculation within `MultiATM` contract is implemented as:

- Maximum fee: 50 basis points (0.5%)
- Fee calculation: Applied to output amount for exact input, input amount for exact output

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 [Info] Lack of Slippage Protection can lead to Unfavorable Exchange Rates

File(s): `MultiATM.sol`

Description: The `MultiATM` contract facilitates token swaps using functions like `swapExactInput(...)` and `swapExactOutput(...)`. These functions determine the final exchanged amount by first calling a view function (e.g., `previewExactInput(...)`), which internally consults the on-chain `Oracle` to get the current price.

The contract does **not** include any mechanism, such as a user-specified minimum output amount or maximum input amount, to protect the user from price fluctuations (slippage) between when the transaction is signed and when it is executed and mined on the blockchain.

The flow for a swap, such as `swapExactInputSingle(...)`, is as follows:

1. The user calls `swapExactInputSingle(input, output, inputAmount, recipient)`.
2. The contract calls `previewExactInputSingle(...)` which uses the oracle price at that moment to calculate `outputAmount`.
3. The contract then calls `_swapExact(...)` using this calculated `outputAmount`.

If the oracle price changes (naturally, due to new data published on the Chainlink feed) between the calculation and the execution of the transaction, the user's effective exchange rate will differ from the rate they initially calculated and expected. Since the contract executes the swap with the calculated amount, the user is forced to accept a potentially much worse rate.

For example, in `swapExactInputSingle(...)`:

```
function swapExactInputSingle(IERC20 input, IERC20 output, uint256 inputAmount, address recipient)
    public
    virtual
    restricted
    returns (
        uint256 /*outputAmount*/
    )
{
    uint256 outputAmount = previewExactInputSingle(input, output, inputAmount);
    // @audit-issue outputAmount is calculated and executed without a user-defined slippage limit.
    _swapExact(input, output, inputAmount, outputAmount, _msgSender(), recipient);
    return outputAmount;
}
```

If the price of `output` token drops (meaning the oracle rate decreases) between the two steps, the user will receive a lower amount of `output` token than expected, leading to an unfavorable exchange.

Recommendation(s): Consider introducing a slippage tolerance parameter to all swap functions. This is a standard security practice in DeFi protocols to protect users from unexpected price changes.

1. For exact input swaps (`swapExactInput(...)` and `swapExactInputSingle(...)`): Add a `minOutputAmount` parameter. The function should revert if the final received amount is less than this user-defined minimum.
2. For exact output swaps (`swapExactOutput(...)` and `swapExactOutputSingle(...)`): Add a `maxInputAmount` parameter. The function should revert if the final required input amount exceeds this user-defined maximum.

Status: Fixed

Update from the client: Fixed `0231aee978de`

6.2 [Info] The contract defaults to 18 decimals if the token does not implement the `decimals(...)` function.

File(s): `MultiATM.sol`

Description: The `tryFetchDecimals(...)` function defaults to 18 decimals if the token contract does not implement the `decimal(...)` function as seen below:

```
function tryFetchDecimals(IERC20 token) view returns (uint8) {
    try IERC20Metadata(address(token)).decimals() returns (uint8 result) {
        return result;
    } catch {
        return 18;
    }
}
```

This function is used when setting token pairs via the `setPair (...)` function, which uses the returned token decimals to set the `numerator` and `denominator` variables, which are used in pricing of tokens during swaps.

As a result, if the default 18 decimals are not accurate as related to the token, this can lead to mispricing of token pairs.

Recommendation(s): Consider reverting if the token does not implement the `decimals (...)` function.

Status: Fixed

Update from the client:Fixed `0231aee978de`

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 Spiko documentation

The Spiko team provided a clear and comprehensive overview of the MultiATM contract during the kick-off call. This was supported with written documentation detailing the different features present in the contracts.

8 Test Suite Evaluation

8.1 Tests Output

```
pnpm compile
> hardhat compile
Compiled 73 Solidity files successfully (evm target: cancun).
pnpm test
> hardhat test

    MultiATM
      stable coin with 6 decimals
         post deployment state
         setFee above max
        with constant price
           preview path rounding
          exact input
            without fees
               preview single
               preview path
               buy token given exact amount of stable - single
               buy token given exact amount of stable - path
               buy stable given exact amount of token - single
               buy stable given exact amount of token - path
               oracle not updated recently (60ms)
            with fees
               preview single
               preview path
               buy token given exact amount of stable - single
               buy token given exact amount of stable - path
               buy stable given exact amount of token - single
               buy stable given exact amount of token - path
               oracle not updated recently (61ms)
          exact output
            without fees
               preview single
               preview path
               buy exact amount of token - single
               buy exact amount of token - path
               buy exact amount of stable - single
               buy exact amount of stable - path
               oracle not updated recently (60ms)
            with fees
               preview single
               preview path
               buy exact amount of token - single
               buy exact amount of token - path
               buy exact amount of stable - single
               buy exact amount of stable - path
               oracle not updated recently (61ms)
        with price increase
           preview path rounding
          exact input
            without fees
               preview single
               preview path
               buy token given exact amount of stable - single
               buy token given exact amount of stable - path
               buy stable given exact amount of token - single
               buy stable given exact amount of token - path
               oracle not updated recently (60ms)
            with fees
               preview single
               preview path
               buy token given exact amount of stable - single
               buy token given exact amount of stable - path
               buy stable given exact amount of token - single
               buy stable given exact amount of token - path
               oracle not updated recently (60ms)
          exact output
```

```
          without fees
             preview single
             preview path
             buy exact amount of token - single
             buy exact amount of token - path
             buy exact amount of stable - single
             buy exact amount of stable - path
             oracle not updated recently (60ms)
          with fees
             preview single
             preview path
             buy exact amount of token - single
             buy exact amount of token - path
             buy exact amount of stable - single
             buy exact amount of stable - path
             oracle not updated recently (59ms)
       with price decrease
          preview path rounding
        exact input
          without fees
             preview single
             preview path
             buy token given exact amount of stable - single
             buy token given exact amount of stable - path
             buy stable given exact amount of token - single
             buy stable given exact amount of token - path
             oracle not updated recently (62ms)
          with fees
             preview single
             preview path
             buy token given exact amount of stable - single
             buy token given exact amount of stable - path
             buy stable given exact amount of token - single
             buy stable given exact amount of token - path
             oracle not updated recently (59ms)
        exact output
          without fees
             preview single
             preview path
             buy exact amount of token - single
             buy exact amount of token - path
             buy exact amount of stable - single
             buy exact amount of stable - path
             oracle not updated recently (57ms)
          with fees
             preview single
             preview path
             buy exact amount of token - single
             buy exact amount of token - path
             buy exact amount of stable - single
             buy exact amount of stable - path
             oracle not updated recently (58ms)
       withdraw
          unauthorized
          partial
          total
    stable coin with 18 decimals
       post deployment state
       setFee above max
       with constant price
          preview path rounding
        exact input
          without fees
             preview single
             preview path
             buy token given exact amount of stable - single
             buy token given exact amount of stable - path
             buy stable given exact amount of token - single
             buy stable given exact amount of token - path
             oracle not updated recently (60ms)
          with fees
             preview single
             preview path
```

```
            buy token given exact amount of stable - single
            buy token given exact amount of stable - path
            buy stable given exact amount of token - single
            buy stable given exact amount of token - path
            oracle not updated recently (61ms)
        exact output
          without fees
            preview single
            preview path
            buy exact amount of token - single
            buy exact amount of token - path
            buy exact amount of stable - single
            buy exact amount of stable - path
            oracle not updated recently (59ms)
          with fees
            preview single
            preview path
            buy exact amount of token - single
            buy exact amount of token - path
            buy exact amount of stable - single
            buy exact amount of stable - path
            oracle not updated recently (58ms)
      with price increase
         preview path rounding
        exact input
          without fees
            preview single
            preview path
            buy token given exact amount of stable - single
            buy token given exact amount of stable - path
            buy stable given exact amount of token - single
            buy stable given exact amount of token - path
            oracle not updated recently (61ms)
          with fees
            preview single
            preview path
            buy token given exact amount of stable - single
            buy token given exact amount of stable - path
            buy stable given exact amount of token - single
            buy stable given exact amount of token - path
            oracle not updated recently (60ms)
        exact output
          without fees
            preview single
            preview path
            buy exact amount of token - single
            buy exact amount of token - path
            buy exact amount of stable - single
            buy exact amount of stable - path
            oracle not updated recently (58ms)
          with fees
            preview single
            preview path
            buy exact amount of token - single
            buy exact amount of token - path
            buy exact amount of stable - single
            buy exact amount of stable - path
            oracle not updated recently (60ms)
      with price decrease
         preview path rounding
        exact input
          without fees
            preview single
            preview path
            buy token given exact amount of stable - single
            buy token given exact amount of stable - path
            buy stable given exact amount of token - single
            buy stable given exact amount of token - path
            oracle not updated recently (58ms)
          with fees
            preview single
            preview path
            buy token given exact amount of stable - single
```

```
            buy token given exact amount of stable - path
            buy stable given exact amount of token - single
            buy stable given exact amount of token - path
            oracle not updated recently (60ms)
        exact output
          without fees
            preview single
            preview path
            buy exact amount of token - single
            buy exact amount of token - path
            buy exact amount of stable - single
            buy exact amount of stable - path
            oracle not updated recently (60ms)
          with fees
            preview single
            preview path
            buy exact amount of token - single
            buy exact amount of token - path
            buy exact amount of stable - single
            buy exact amount of stable - path
            oracle not updated recently (61ms)
      withdraw
         unauthorized
         partial
         total
    stable coin with 36 decimals
       post deployment state
       setFee above max
      with constant price
         preview path rounding
        exact input
          without fees
            preview single
            preview path
            buy token given exact amount of stable - single
            buy token given exact amount of stable - path
            buy stable given exact amount of token - single
            buy stable given exact amount of token - path
            oracle not updated recently (59ms)
          with fees
            preview single
            preview path
            buy token given exact amount of stable - single
            buy token given exact amount of stable - path
            buy stable given exact amount of token - single
            buy stable given exact amount of token - path
            oracle not updated recently (60ms)
        exact output
          without fees
            preview single
            preview path
            buy exact amount of token - single
            buy exact amount of token - path
            buy exact amount of stable - single
            buy exact amount of stable - path
            oracle not updated recently (60ms)
          with fees
            preview single
            preview path
            buy exact amount of token - single
            buy exact amount of token - path
            buy exact amount of stable - single
            buy exact amount of stable - path
            oracle not updated recently (62ms)
      with price increase
         preview path rounding
        exact input
          without fees
            preview single
            preview path
            buy token given exact amount of stable - single
            buy token given exact amount of stable - path
            buy stable given exact amount of token - single
```

```
            buy stable given exact amount of token - path
            oracle not updated recently (58ms)
         with fees
            preview single
            preview path
            buy token given exact amount of stable - single
            buy token given exact amount of stable - path
            buy stable given exact amount of token - single
            buy stable given exact amount of token - path
            oracle not updated recently (60ms)
      exact output
         without fees
            preview single
            preview path
            buy exact amount of token - single
            buy exact amount of token - path
            buy exact amount of stable - single
            buy exact amount of stable - path
            oracle not updated recently (61ms)
         with fees
            preview single
            preview path
            buy exact amount of token - single
            buy exact amount of token - path
            buy exact amount of stable - single
            buy exact amount of stable - path
            oracle not updated recently (59ms)
   with price decrease
      preview path rounding
      exact input
         without fees
            preview single
            preview path
            buy token given exact amount of stable - single
            buy token given exact amount of stable - path
            buy stable given exact amount of token - single
            buy stable given exact amount of token - path
            oracle not updated recently (60ms)
         with fees
            preview single
            preview path
            buy token given exact amount of stable - single
            buy token given exact amount of stable - path
            buy stable given exact amount of token - single
            buy stable given exact amount of token - path
            oracle not updated recently (59ms)
      exact output
         without fees
            preview single
            preview path
            buy exact amount of token - single
            buy exact amount of token - path
            buy exact amount of stable - single
            buy exact amount of stable - path
            oracle not updated recently (58ms)
         with fees
            preview single
            preview path
            buy exact amount of token - single
            buy exact amount of token - path
            buy exact amount of stable - single
            buy exact amount of stable - path
            oracle not updated recently (57ms)
   withdraw
      unauthorized
      partial
      total
```

8.2 Automated Tools

8.2.1 AuditAgent

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

9 About Nethermind

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

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

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

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

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

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

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

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

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

General Advisory to Clients

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

Disclaimer

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