Security Review Report
NM-0735 - MELLOW - SWAP MODULE SECURITY REVIEW

NETHERMIND
SECURITY

(November 19, 2025)

Contents

1 Executive Summary

This document presents the security review performed by Nethermind Security for Mellow Protocol smart contracts. The scope of this audit was specifically focused on the newly developed `SwapModule` contract.

The **Mellow Protocol's `SwapModule`** is a smart contract designed to provide secure, flexible, and permissioned trade execution for the protocol's vaults (referred to as `subvaults`). It acts as a specialized utility layer that isolates trading operations from the core vault logic.

The module's primary function is to execute token swaps on behalf of a `subvault` through two distinct mechanisms. First, it facilitates immediate, on-chain swaps by interfacing with a whitelist of approved external DEX aggregators and routers. Second, it supports gasless, off-chain limit orders through a native integration with CoW Protocol, allowing trades to be settled by off-chain solvers when specific price targets are met.

The audit comprises 244 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, all of which are classified as 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 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.

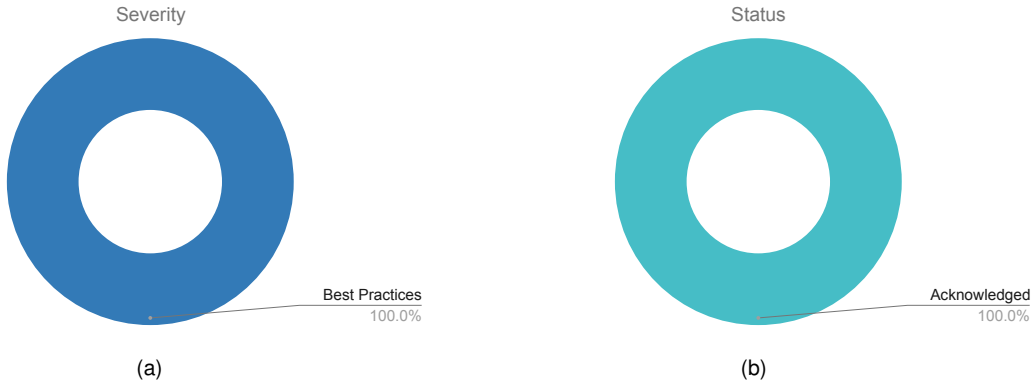

Fig. 1: Distribution of issues: Critical (0), **High** (0), **Medium** (0), **Low** (0), **Undetermined** (0), **Informational** (0), **Best Practices** (2). **Distribution of status: Fixed** (0), **Acknowledged** (2), **Mitigated** (0), **Unresolved** (0)

Summary of the Audit

Audit Type	Security Review
Initial Report	November 19, 2025
Final Report	November 19, 2025
Initial Commit	688382e
Final Commit	688382e
Documentation Assessment	Low
Test Suite Assessment	High

2 Audited Files

	Contract	LoC	Comments	Ratio	Blank	Total
1	src/utils/SwapModule.sol	244	33	13.5%	35	312
	Total	**244**	**33**	**13.5%**	**35**	**312**

3 Summary of Issues

	Finding	Severity	Update
1	The `checkParams(...)` function does not check for zero `minAmountOut`	Best Practices	Acknowledged
2	The `initialize(...)` function lacks a check for equal array lengths of `holders` and `roles`	Best Practices	Acknowledged

4 System Overview

For a comprehensive understanding of the entire Mellow Protocol system, please refer to the official audit report: NM0587-FINAL_Mellow. This section focuses exclusively on the newly added SwapModule contract.

The Mellow Protocol's SwapModule is a permissioned utility contract designed to execute token swaps on behalf of a designated subvault contract. It functions as a secure swapping layer, enabling the protocol to trade assets through two distinct mechanisms: immediate on-chain transactions via whitelisted external routers, and gasless off-chain limit orders using CoW Protocol. The module's design emphasizes security through a strict, role-based access control system and mandatory on-chain oracle price checks for slippage protection.

4.1 Access Control and Roles

The contract is built upon the MellowACL (Access Control List) system, which defines a set of specific roles to manage permissions. All primary functions are restricted to addresses holding these roles, ensuring that only authorized entities can perform sensitive operations.

- **subvault**: This is the address of the associated vault contract that "owns" the assets. The subvault is the *only* entity permitted to push assets into or pull assets from the SwapModule.
- **CALLER_ROLE**: This role is granted to trusted operators, such as off-chain keepers or managers. Holders of this role are responsible for initiating all trading operations, including both on-chain swaps (swap()) and the creation or invalidation of CoW Protocol orders.
- **ROUTER_ROLE**: This serves as a whitelist for external smart contracts (e.g., Uniswap V3's router, 1inch's aggregator) that are permitted to be used for executing on-chain swaps.
- **TOKEN_IN_ROLE** / **TOKEN_OUT_ROLE**: These roles create whitelists for assets that are approved for trading through the module.
- **SET_SLIPPAGE_ROLE**: This administrative role has permission to set the defaultMultiplier and customMultiplier values, which control the acceptable slippage tolerance for all trades.
- **DEFAULT_ADMIN_ROLE**: This role has full administrative privileges to manage all other roles and can update critical parameters like the oracle address.

4.2 Asset Management Flow

The SwapModule is designed as a transitory contract. The movement of assets is strictly controlled by the subvault contract.

- **Pushing Assets**: Before a swap can be initiated, the subvault must first transfer the required tokenIn to the SwapModule by calling the pushAssets() function. This function ensures that only the designated subvault can fund the module for a trade.
- **Pulling Assets**: After a swap is completed, the resulting tokenOut (or any remaining tokenIn) is held by the SwapModule. The subvault can then call pullAssets() to retrieve these tokens.

4.3 On-Chain Swap Execution

For immediate, on-chain trades, an address with the CALLER_ROLE invokes the swap() function. This flow interacts directly with a whitelisted external DEX aggregator or router.

The process is as follows:

- **1. Initiation**: A CALLER_ROLE submits the trade details via the Params struct, which includes tokenIn, tokenOut, amountIn, minAmountOut, and a deadline. The call must also specify a router address and the data for the external call.
- **2. Validation**: The swap() function first performs a series of checks via checkParams(). It verifies that the tokens are whitelisted, the deadline is valid, and the module holds sufficient amountIn.
- **3. Oracle Price Check**: checkParams() queries the configured IAaveOracle (via the evaluate() function) to get the current market price of tokenIn versus tokenOut. It then applies the configured multiplier (slippage tolerance) to calculate an oracleMinAmount. The transaction is reverted if the caller's specified minAmountOut is less than this oracle-derived minimum, protecting against poor execution or oracle deviation.
- **4. Execution**: If all checks pass, the module verifies the router has the ROUTER_ROLE. It then approves the router to spend the amountIn and executes the swap by calling the router with the provided data.
- **5. Final Check**: After the external call, the swap() function verifies that the amountOut of tokens received is greater than or equal to the minAmountOut.

4.4 CoW Protocol Limit Order Flow

As an alternative to immediate on-chain swaps, the module can create gasless, off-chain limit orders using CoW Protocol (GPv2Settlement).

This flow allows the protocol to set a specific price target without committing gas upfront.

- **1. Creation**: A CALLER_ROLE calls createLimitOrder(), providing the Params struct, a GPv2Order.Data struct, and a unique orderUid.

- **2. Validation**: The function performs the same critical `checkParams()` validation as an on-chain swap, including the oracle-based minimum output check. It then uses `checkCowswapOrder()` to ensure all parameters in the CoW Protocol `order` (e.g., tokens, amounts, deadlines) are consistent with the `Params`.
- **3. Pre-Signature**: Upon successful validation, the `SwapModule` calls `setPreSignature(orderUid, true)` on the `GPv2Settlement` contract. This "pre-signs" the order on-chain, authorizing CoW Protocol's solvers to execute the swap on the module's behalf if and when the order's limit price (`order.buyAmount`) is met.
- **4. Execution**: The order is executed off-chain by CoW Protocol's solvers. The `cowswapVaultRelayer` (which must be approved via `setCowswapApproval()`) pulls the `tokenIn` from the SwapModule and delivers the `tokenOut` upon successful settlement.
- **5. Invalidation**: A `CALLER_ROLE` can also call `invalidateOrder()` at any time to cancel a pending pre-signed order.

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 [Best Practices] The `checkParams(...)` function does not check for zero `minAmountOut`

File(s): src/utils/SwapModule.sol

Description: The `checkParams(...)` function is called by both `swap(...)` and `createLimitOrder(...)` to validate swap parameters before execution. This function performs several important sanity checks, including correctly verifying that `params.amountIn` is not `0`.

However, the function fails to implement a similar check to ensure that `params.minAmountOut` is also greater than `0`. This makes it possible for a swap or limit order to be processed with an expected minimum output of `0`.

This is particularly relevant for `createLimitOrder(...)`, which creates CoW Protocol orders. In this context, `params.minAmountOut` maps to the order's `buyAmount`. The CoW Protocol documentation explicitly advises against signing orders where `buyAmount` is `0`.

```
// src/utils/SwapModule.sol
function checkParams(Params calldata params) public view {
// ...
// @audit This check correctly ensures amountIn is not zero.
    if (TransferLibrary.balanceOf(params.tokenIn, address(this)) < params.amountIn || params.amountIn == 0) {
        revert Forbidden("amountIn");
    }
    SwapModuleStorage storage $ = _swapModuleStorage();
// ...
    uint256 oracleMinAmount =
        Math.mulDiv(
            evaluate(params.tokenIn, params.tokenOut, params.amountIn),
            multiplier,
            BASE_MULTIPLIER
        );

    // @audit-issue A check to ensure `params.minAmountOut > 0` is missing.
    if (params.minAmountOut < oracleMinAmount) {
        revert Forbidden("minAmountOut < oracleMinAmount");
    }
    if (params.deadline < block.timestamp) {
// ...
    }
}
```

Recommendation(s): Consider adding a validation check in the `checkParams(...)` function to ensure that `params.minAmountOut` is strictly greater than `0`.

Status: Acknowledged

Update from the client:

6.2 [Best Practices] The `initialize(...)` function lacks a check for equal array lengths of `holders` and `roles`

File(s): src/utils/SwapModule.sol

Description: The `initialize(...)` function in the `SwapModule` contract decodes initialization parameters from a `bytes calldata data` argument. This includes a `holders` array and a `roles` array, which are then used to grant permissions by looping over them.

The problem is that the function does not validate that `holders.length` is equal to `roles.length`. The loop iterates based on `holders.length`.

This means that if an administrator provides arrays of mismatched lengths (e.g., `holders.length` is less than `roles.length`) due to an off-chain error, the transaction will succeed silently. It will only grant roles up to the shorter `holders.length`, leading to a partially configured contract where expected permissions are not set. This violates the "fail-fast" principle and can make debugging misconfigurations difficult.

```
/// @inheritdoc IFactoryEntity
function initialize(bytes calldata data) external initializer {
    (
        address admin,
        address subvault_,
        address oracle_,
        uint256 defaultMultiplier_,
        address[] memory holders,
        bytes32[] memory roles
    ) = abi.decode(data, (address, address, address, uint256, address[], bytes32[]));
    if (admin == address(0) || subvault_ == address(0) || oracle_ == address(0)) {
        revert ZeroValue();
    }
    checkMultiplier(defaultMultiplier_);
    SwapModuleStorage storage $ = _swapModuleStorage();

    _grantRole(DEFAULT_ADMIN_ROLE, admin);
    $.subvault = subvault_;
    $.oracle = oracle_;
    $.defaultMultiplier = defaultMultiplier_;
    // @audit-issue This loop iterates based on `holders.length` but does not
    // @audit-issue validate that `holders.length` equals `roles.length`.
    for (uint256 i = 0; i < holders.length; i++) {
        if (holders[i] == address(0) || roles[i] == bytes32(0)) {
            revert ZeroValue();
        }
        _grantRole(roles[i], holders[i]);
    }
    emit Initialized(data);
}
```

Recommendation(s): Consider adding a validation check at the beginning of the `initialize(...)` function to require that `holders.length == roles.length`. The function should revert if the lengths are mismatched.

Status: Acknowledged

Update from the client:

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 the `SwapModule` documentation

No documentation for the `SwapModule` contract was provided, but Mellow's team addressed all the questions and concerns raised by the Nethermind Security team, providing valuable insights and a comprehensive understanding of the project's technical aspects.

8 Test Suite Evaluation

8.1 Compilation Output

```
forge build --use 0.8.25 --cache-path cache

[] Compiling...
[] Compiling 275 files with Solc 0.8.25
[] Solc 0.8.25 finished in 60.70s
Compiler run successful!
Done in 61.13s.
```

8.2 Tests Output

```
forge test --match-contract "SwapModule" --fork-url $ETH_RPC_URL --gas-limit 10000000000000000 --fork-block-number
↪  23437454

Ran 9 tests for test/unit/utils/SwapModule.t.sol:SwapModuleTest
[PASS] testCheckCowswapOrder_NO_CI() (gas: 2013862)
[PASS] testConstructor_NO_CI() (gas: 25807)
[PASS] testCreateCowswapOrder_NO_CI() (gas: 2385352)
[PASS] testInitializer_NO_CI() (gas: 2612457)
[PASS] testSetters_NO_CI() (gas: 2087590)
[PASS] testSingleSwap_ETH_WETH_NO_CI() (gas: 2131264)
[PASS] testSingleSwap_USDC_USDT_NO_CI() (gas: 3315727)
[PASS] testSingleSwap_USDC_WETH_NO_CI() (gas: 3102775)
[PASS] testSingleSwap_WETH_USDT_NO_CI() (gas: 2957530)
Suite result: ok. 9 passed; 0 failed; 0 skipped; finished in 10.52s (33.14s CPU time)

Ran 1 test for test/integration/SwapModuleTest.t.sol:SwapModuleIntegration
[PASS] testSwapModule_Integration_NO_CI() (gas: 13857758)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 28.24s (12.05s CPU time)

Ran 2 test suites in 28.47s (38.76s CPU time): 10 tests passed, 0 failed, 0 skipped (10 total tests)
```

9 About Nethermind

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

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

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

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

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

Cryptography Research: At Nethermind, our cryptography Research team 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.