Security Review Report
NM-0296 Dinero

NETHERMIND
SECURITY

(Sep 9, 2024)

Contents

1 Executive Summary

This document presents the security review performed by Nethermind Security for the Dinero. This security engagement focused on two areas: the fix review of issues found in the Branded LST contracts by a different audit provider and a security review of a new feature enabling the deployment of the Branded LST contracts on the Arbitrum network. The deployment script of the new Arbitrum contracts was also in the scope of this security engagement.

The audit was performed using (a) manual analysis of the codebase, (b) automated analysis tools, (c) simulation of the smart contract, and (d) creation of test cases. **Along this document, we report** two points of attention, where one is classified as `Undetermined` and one is classified as `Best Practice`. 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 concludes the document.

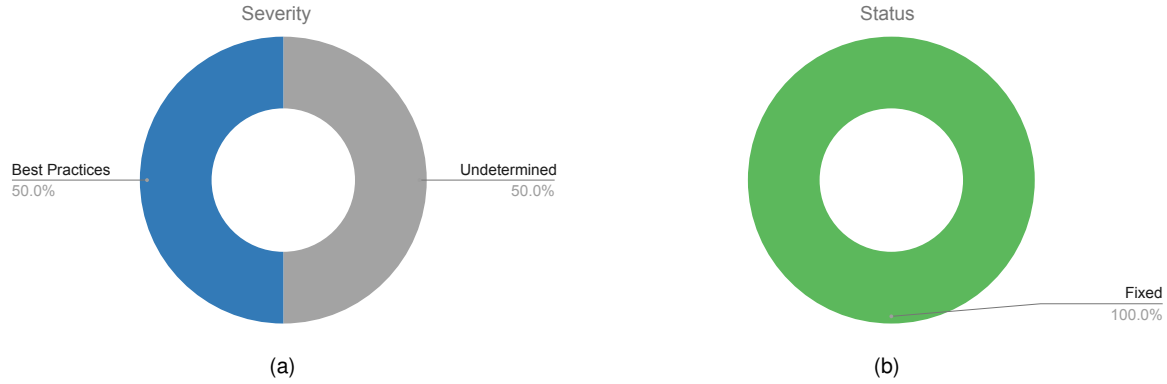

Fig. 1: Distribution of issues: Critical (0), **High** (0), **Medium** (0), **Low** (0), **Undetermined** (1), **Informational** (0), **Best Practices** (1). **Distribution of status: Fixed** (2), **Acknowledged** (0), **Mitigated** (0), **Unresolved** (0)

Summary of the Audit

Audit Type	Security Review
Initial Report	Sep 6, 2024
Response from Client	Regular responses during audit engagement
Final Report	Sep 9, 2024
Repository	dinero-pirex-eth
Commit (Audit)	c716465f06b6838c57c8d7d2fc918c1918bfb4ba
Commit (Final)	66fb6b279032908f2c8b3549aec6f63a6c6d9309
Documentation Assessment	High
Test Suite Assessment	Low

2 Audited Files

The following files were reviewed as part of the new Arbitrum deployment.

	Contract	LoC	Comments	Ratio	Blank	Total
1	layer2/L1ArbReceiverETH.sol	40	19	47.5%	7	66
2	layer2/LiquidStakingTokenArb.sol	69	32	46.4%	14	115
	Total	**109**	**51**	**46.8%**	**21**	**181**

The following files were part of a diff audit between the commits d463e08 and 9bf194b.

	Contract	LoC	Comments	Ratio	Blank	Total
1	layer2/DineroERC20RebaseUpgradeable.sol	179	206	115.1%	55	440
2	layer2/L2ExchangeRateProvider.sol	83	25	30.1%	23	131
3	layer2/L2SyncPool.sol	223	156	70.0%	54	433
4	layer2/LiquidStakingTokenLockbox.sol	469	251	53.5%	139	859
5	layer2/L1SyncPool.sol	142	137	96.5%	48	327
6	layer2/LiquidStakingToken.sol	526	342	65.0%	134	1002
	Total	**1622**	**1117**	**68.9%**	**453**	**3192**

3 Summary of Issues

	Finding	Severity	Update
1	Contract calls to the `l2ToL1Sender(...)` function on the Arbitrum's `Bridge` contract will always revert	Undetermined	Fixed
2	Token addresses are not validated in the `_sync(...)` function	Best Practices	Fixed

4 System Overview

The `LiquidStakingToken` contract is the extension of the `Pirex` protocol for the Layer 2 (L2) chains. Instead of holding the `AutoPxEth` vault shares on Layer 1 (L1), the users can hold the Liquid Staking Token (LST) on L2 and benefit from the same `apxETH` yield. The LST can be obtained by performing a deposit on either L1 or L2. On L1, the accepted tokens include `Ether`, `pxETH`, and `apxETH`, while on L2, only the Layer 2 `Ether` is accepted. Unlike deposits, the withdrawals can be initiated exclusively from the L2 side. In exchange for the LST, users receive an appropriate amount of `pxETH` tokens on L1 based on the conversion ratio from the `AutoPxEth` vault.

The following sections delve into the system's components and their interactions. The diagram below showcases a high-level view of the system's architecture.

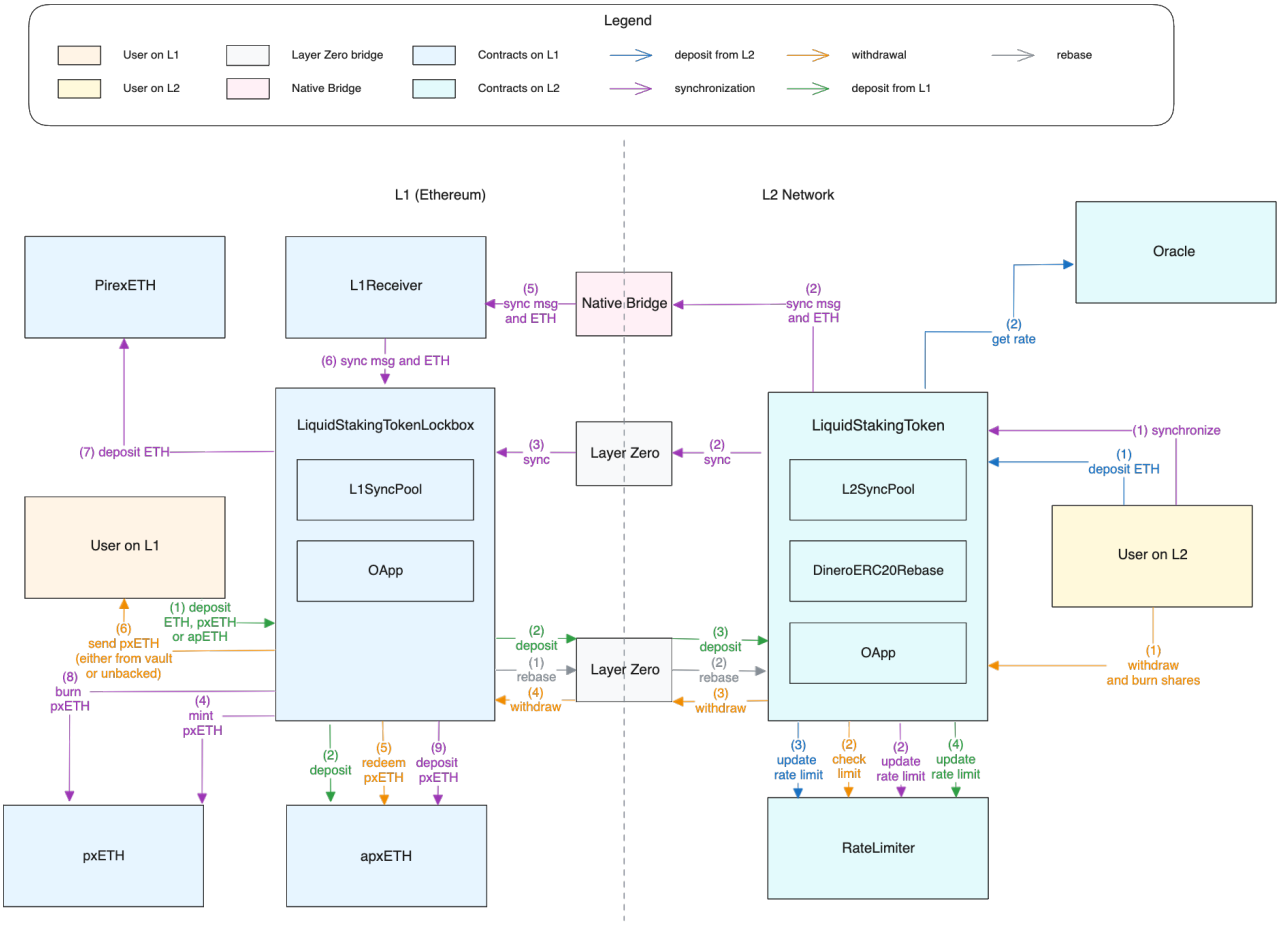

Fig. 2: Dinero overview.

4.1 Deposits

Layer 1 deposits are facilitated through the `LiquidStakingTokenLockbox` contract. The Lockbox is responsible for handling the `Ether`, `pxETH`, and `apxETH` token deposits into the `Pirex` protocol. Communication between L1 and L2 is possible using the LayerZero messaging system. During the deposit, the Lockbox sends the LayerZero message to the Layer 2 chain with the information about the L1 state changes. The `LiquidStakingToken` contract that receives this message on L2 can use the provided information to mint an appropriate amount of LST shares as well as perform a rebase using the new assets per share ratio from the `AutoPxEth` vault.

Layer 2 deposits are facilitated through the `LiquidStakingToken` contract. Since the L2 Ether cannot directly be used for staking in the `Pirex` protocol, it must be first bridged to the L1 `Ether` via the native Layer 2 bridge. In the case of the Mode network, on which the project will be deployed first, it is the Standard Bridge from the Optimism Stack. For the Arbitrum deployment, it is the Arbitrum's Native Bridge. The L2 deposits are batched together and sent to L1 during the cross-chain syncing process. The synchronization mechanism itself is explained in greater detail in the following section. Once `Ether` is released from the L1 native bridge, it is deposited into the `Pirex` protocol to start the `Ether` staking process. The validator staking rewards generate the yield for the LST and `apxETH` token holders.

4.2 Cross-chain synchronization

The L1 chain is unaware of the user deposits on L2 until the two chains are synchronized. The syncing process can be done by calling the `sync(...)` function from the `L2SyncPool` contract. To keep the chain states up to date, the off-chain keeper will trigger the synchronization regularly. This action can only be performed by the Dinero's Sync Keeper once a certain threshold of deposits is reached.

The synchronization mechanism is split into two parts: the slow sync and the fast sync. **The slow sync** process sends the native Layer 2 tokens to Layer 1 over the native bridge. Due to the nature of optimistic rollups and the design of the fault-proof system, this process takes at least 7 days to finalize. The message won't be relayed during that time, and the Ether won't be released on L1. To mitigate this limitation, **the fast sync** message is sent via LayerZero omnichain messaging protocol to inform the L1 about the deposit on L2. The `LiquidStakingTokenLockbox` contract receives this message and mints `pxETH` tokens in anticipation of the `Ether` that is yet to be released from the bridge. The newly minted `pxETH` tokens await the slow sync process to finish inside the `LiquidStakingTokenLockbox` contract. If, during the waiting period, users request withdrawals on L2, whenever possible, they will be provided with the `pxETH` tokens from the Lockbox first instead of withdrawing funds from the `AutoPxEth` vault. The fast sync mechanism enables immediate liquidity for the L2 users without affecting the existing `AutoPxEth` deposits before the actual Ether arrives from the bridge.

4.3 Withdrawals

Unlike deposits, withdrawals can only be initiated on the L2 side. Users can call the `withdraw(...)` function on the `LiquidStakingToken` contract and specify the amount of assets that they want to withdraw. Their LST shares will be burned on L2, and a withdrawal message will be sent to L1 via LayerZero. Once the `LiquidStakingTokenLockbox` contract receives the message, it will transfer the `pxETH` tokens to the user on L1. As mentioned in the previous section, the Lockbox will first attempt to use all the `pxETH` that it currently holds, and only after that will it start withdrawing additional `pxETH` tokens from the vault.

4.4 Rebases

Whenever new rewards are distributed into the `AutoPxEth` vault, the price of an individual `apxETH` share increases. Since LST shares are L2 representations of the `apxETH` shares, this share price increase must also be reflected on the Layer 2 chain. The rebase mechanism informs the L2 about the newest assets per share ratio from the `AutoPxEth` vault. The current L1 share price is used to update the internal accounting on L2. Similarly to the synchronization mechanism, calling `rebase(...)` is permissionless but will be regularly called by the Keeper.

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 [Undetermined] Contract calls to the `l2ToL1Sender(...)` function on the Arbitrum's `Bridge` contract will always revert

File(s): `L1ArbReceiverETH.sol`

Description: Whenever the `L2` sync message is received in the `L1` Abitrum's `Outbox` contract, it must be executed by Dinero's Keeper. This action is performed by calling the `executeTransaction(...)` function on the `Outbbox` contract. The `Outbox` contract, in turn, calls the Arbitrum's `Bridge` contract to execute the call with the data sent from the `L2`. The target of this call is the `onMessageReceived(...)` function in the `L1ArbReceiverETH` contract, where the message is decoded and processed.

Before the data is forwarded to the `L1SyncPool`, the `_forwardToL1SyncPool(...)` checks that the current `msg.sender` is the Arbitrum's `Bridge` contract, and the `sender` that sent the transaction on `L2` is an authorized contract - the `LiquidStakingTokenArb` contract.

```
function _forwardToL1SyncPool(
    // ...
    bytes32 sender, // l2toL1Sender
    // ...
) internal virtual {
    // ...
    // @audit The L1 caller should be the bridge
    if (msg.sender != getMessenger()) revert L1BaseReceiver__UnauthorizedCaller();
    // @audit The L2 caller should be the LiquidStakingTokenArb contract
    if (_getAuthorizedL2Address(originEid) != sender) revert L1BaseReceiver__UnauthorizedL2Sender();
}
```

The `L2` sender address is retrieved by calling the `l2ToL1Sender(...)` function on a contract address returned from a call to `getMessenger(...)`.

```
// @audit The `getMessenger(...)` returns the L1Messenger address,
// which is set to Arbitrum's Bridge in deployment scripts
address sender = IArbitrumMessenger(getMessenger()).l2ToL1Sender();
```

The problem is that the current deployment setup sets the `L1Messenger` address to point to the Arbitrum's `Bridge` contract, which does not have the `l2ToL1Sender(...)` function defined. As a result, every call to the `onMessageReceived(...)` function would result in a revert. Because of this, the slow sync process couldn't be finalized until the `L1ArbReceiverEth` contract was updated to fix the issue. Since technically no user funds are at risk, but the project won't work in the current configuration we asses the severity to be undetermined.

Recommendation(s): The `l2ToL1Sender` address can be accessed from the `Outbox` contract instead. Consider including the `Outbox` contract address in the `ARBITRUM` deployment configuration so that the `l2ToL1Sender` address can be retrieved successfully.

Status: Fixed

6.2 [Best Practices] Token addresses are not validated in the `_sync(...)` function

File(s): `LiquidStakingTokenArb.sol`

Description: The `_sync(...)` function in the `LiquidStakingTokenArb` contract sends fast and slow sync messages from `L2` to `L1`. Whenever handling cross-chain messages, it is considered a best practice to implement restrictive input validation checks on the sender's side. This prevents a situation in which the message is sent successfully from one chain, but it reverts on the receiving chain, causing the chain states to go out of sync.

The described scenario can occur in the `_sync(...)` function since it does not check the provided `L1` and `L2` token addresses, but such checks are present in the receiving contract - `L1ArbReceiverEth`. Both the receiver contract and the `LiquidStakingTokenArb` are expected to work with Ether only. If, for some reason, any other tokens are added with the `setL1TokenIn(...)` function, the contracts won't work properly.

Recommendation(s): Consider validating that the `_l1TokenIn` and `_l2TokenIn` addresses match the `Constants.ETH_ADDRESS`.

Status: Fixed

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 Worldcoin documentation

The documentation for the Dinero Pirex protocol is contained in the project's GitHub readme file. It provides a high-level overview of the project and contains helpful diagrams visualising the contract flows. **The team answered every question during meetings or through messages**, which gave the auditing team a lot of insight and a deep understanding of the technical aspects of the project.

8 Test Suite Evaluation

```
./src/scripts/forgeTest.sh
[] Compiling...
No files changed, compilation skipped

Ran 23 tests for test/ModeEth.t.sol:L2TestModeEth
[PASS] testDepositApxEth() (gas: 520728)
[PASS] testDepositEth() (gas: 509334)
[PASS] testDepositPxEth() (gas: 570399)
[PASS] testLiquidStakingTokenInitialization() (gas: 41825)
[PASS] testLiquidStakingTokenLockboxInitialization() (gas: 12975)
[PASS] testLzReceive() (gas: 673446)
[PASS] testLzReceiveDeposit() (gas: 192094)
[PASS] testLzReceiveRebase() (gas: 722543)
[PASS] testLzReceiveWithdrawDecreaseAssetsPerShare() (gas: 1103565)
[PASS] testLzReceiveWithdrawIncreasedAssetsPerShare() (gas: 1077015)
[PASS] testQuoteDeposit() (gas: 145330)
[PASS] testQuoteRebase() (gas: 145103)
[PASS] testQuoteWithdraw() (gas: 148592)
[PASS] testRebase() (gas: 773878)
[PASS] testRevertLiquidStakingTokenLockboxLzReceive_NotAllowed() (gas: 62192)
[PASS] testRevertSetRebaseFee_InvalidFee() (gas: 18279)
[PASS] testRevertSetTreasury_ZeroAddress() (gas: 18235)
[PASS] testRevertWithdrawLiquidStakingToken_ZeroAddress() (gas: 25601)
[PASS] testRevertWithdrawLiquidStakingToken_ZeroAmount() (gas: 27708)
[PASS] testSetTreasury() (gas: 26495)
[PASS] testSetTreasuryFee() (gas: 24573)
[PASS] testWithdrawLiquidStakingToken() (gas: 884596)
[PASS] test_depositEth_rebase_modeEthWithdraw() (gas: 1718057)
Suite result: ok. 23 passed; 0 failed; 0 skipped; finished in 24.47s (52.52s CPU time)

Ran 9 tests for test/ModeEthSyncPool.t.sol:L2TestModeEthSyncPool
[PASS] test_Deposit() (gas: 1237877)
[PASS] test_Sync() (gas: 639571)
[PASS] test_SyncAndFinalize() (gas: 691168)
[PASS] test_SyncAndFinalizeMultiple() (gas: 1241501)
[PASS] test_SyncAndFinalizeMultipleUnordered() (gas: 1241938)
[PASS] test_SyncAndFinalizeMultipleUnorderedIncomplete() (gas: 1247239)
[PASS] test_SyncAndWithdraw() (gas: 781386)
[PASS] test_SyncWithdrawAndFinalize() (gas: 845696)
[PASS] test_SyncWithdrawAndFinalizeMultipleUnordered() (gas: 1103452)
Suite result: ok. 9 passed; 0 failed; 0 skipped; finished in 28.14s (51.08s CPU time)

Ran 2 test suites in 28.15s (52.61s CPU time): 32 tests passed, 0 failed, 0 skipped (32 total tests)
[] Compiling...
No files changed, compilation skipped

Ran 9 tests for test/ArbEthSyncPool.t.sol:L2TestArbEthSyncPool
[PASS] test_Deposit() (gas: 1100542)
[PASS] test_Sync() (gas: 574887)
[PASS] test_SyncAndFinalize() (gas: 626418)
[PASS] test_SyncAndFinalizeMultiple() (gas: 1108164)
[PASS] test_SyncAndFinalizeMultipleUnordered() (gas: 1108600)
[PASS] test_SyncAndFinalizeMultipleUnorderedIncomplete() (gas: 1105521)
[PASS] test_SyncAndWithdraw() (gas: 716636)
[PASS] test_SyncWithdrawAndFinalize() (gas: 780946)
[PASS] test_SyncWithdrawAndFinalizeMultipleUnordered() (gas: 987730)
Suite result: ok. 9 passed; 0 failed; 0 skipped; finished in 23.12s (38.17s CPU time)

Ran 1 test suite in 23.13s (23.12s CPU time): 9 tests passed, 0 failed, 0 skipped (9 total tests)
error: a value is required for '--fork-url <URL>' but none was supplied

For more information, try '--help'.
```

9 About Nethermind

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

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

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

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

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

Cryptography Research: At Nethermind, our Cryptography Research team is dedicated to continuous internal research while fostering close collaboration with external partners. The team has expertise across a wide range of domains, including cryptography protocols, consensus design, decentralized identity, verifiable credentials, Sybil resistance, oracles, and credentials, distributed validator technology (DVT), and Zero-knowledge proofs. This diverse skill set, combined with strong collaboration between our engineering teams, enables us to deliver cutting-edge solutions to our partners and clients.

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

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

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

Learn more about us at nethermind.io.

General Advisory to Clients

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

Disclaimer

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