Security Review Report
NM-0144 WORLDCOIN

WorldID Contracts Upgrade

NETHERMIND

SECURITY

(Oct 12, 2023)

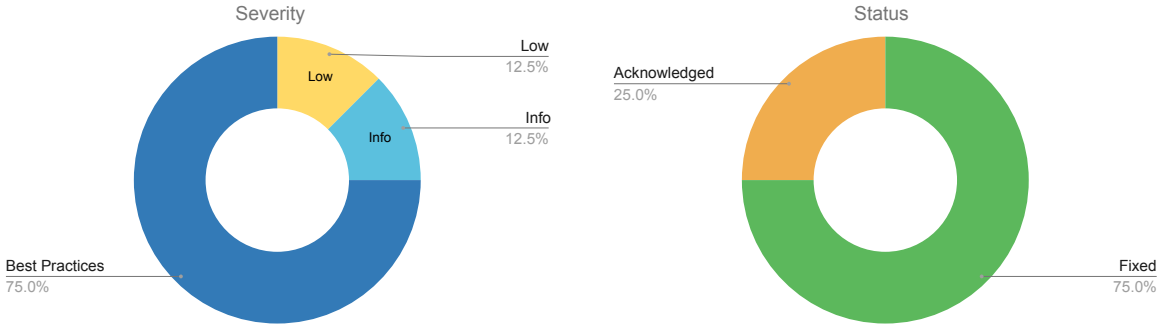
Contents

1 Executive Summary

This document presents the security review performed by Nethermind on the WorldID Contracts Upgrade containing 449 lines of code. The Worldcoin team has implemented a new feature that allows the removal of identities from the tree structure. To implement this feature, a new contract called `WorldIDIdentityManagerImplV2` has been created. This contract inherits from the v1 implementation and is responsible for receiving an array of identities to be deleted, verifying their proof, and updating the latest root accordingly.

In addition, there has been an update to the code where all elements now come out reduced from the circuit. As a result, all checks for inputs being reduced have been removed from the contract. **Along the audit of this contract, we report** 8 points of attention, where 1 is classified as `Low`, 1 is classified as `Info` and 6 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 of this audit. Section 3 summarizes the issues. Section 4 discusses the risk rating methodology adopted for this audit. Section 5 details the issues. Section 6 discusses the documentation provided by the client for this audit. Section 7 presents the compilation, tests, and automated tests. Section 8 concludes the document.



(a) distribution of issues

(b) status of issues

Fig. 1: Distribution of issues: Critical (0), **High** (0), **Medium** (0), **Low** (1), **Undetermined** (0), **Informational** (1), **Best Practices** (6). **Distribution of status: Fixed** (6), **Acknowledged** (2), **Mitigated** (0), **Unresolved** (0), **Partially Fixed** (0)

Summary of the Audit

Audit Type	Security Review
Initial Report	Oct. 6, 2023
Response from Client	Oct. 10, 2023
Final Report	Oct. 12, 2023
Methods	Manual Review, Automated Analysis
Repository	world-id-contracts
Commit Hash (Audit)	42c26ecbd82fba56983addee6485d5b617960a2a
Documentation	README.md
Documentation Assessment	High
Test Suite Assessment	High

2 Audited Files

	Contract	LoC	Comments	Ratio	Blank	Total
1	WorldIDIdentityManager.sol	6	24	400.0%	4	34
2	WorldIDIdentityManagerImplV2.sol	71	122	171.8%	19	212
3	WorldIDIdentityManagerImplV1.sol	269	325	120.8%	86	680
4	abstract/WorldIDProxy.sol	6	19	316.7%	3	28
5	abstract/WorldIDImpl.sol	22	56	254.5%	11	89
6	utils/CheckInitialized.sol	16	17	106.2%	6	39
7	utils/UnimplementedTreeVerifier.sol	10	19	190.0%	3	32
8	utils/SemaphoreTreeDepthValidator.sol	8	8	100.0%	1	17
9	interfaces/IWorldID.sol	11	16	145.5%	2	29
10	interfaces/IBridge.sol	5	7	140.0%	2	14
11	interfaces/IWorldIDGroups.sol	12	18	150.0%	2	32
12	interfaces/ITreeVerifier.sol	4	13	325.0%	1	18
13	interfaces/ISemaphoreVerifier.sol	4	13	325.0%	1	18
14	interfaces/IBaseWorldID.sol	5	10	200.0%	3	18
	Total	**449**	**667**	**148.6%**	**144**	**1260**

3 Summary of Issues

	Finding	Severity	Update
1	Invalid `batchSize` can result in incorrect hash calculation	Low	Fixed
2	Possible duplicate roots block propagating root to L2 contracts	Info	Acknowledged
3	Absence of event emission during contract initialization	Best Practices	Fixed
4	Floating pragma	Best Practices	Acknowledged
5	Function `deleteIdentities(...)` has different access control than stated in the documentation	Best Practices	Fixed
6	Missing input validation in some functions	Best Practices	Fixed
7	Outdated NatSpec documentation and comments in V2 implementation	Best Practices	Fixed
8	Used event marked as unused in the documentation	Best Practices	Fixed

4 Risk Rating Methodology

The risk rating methodology used by Nethermind follows the principles established by the OWASP Foundation. The severity of each finding is determined by two factors: **Likelihood** and **Impact**.

Likelihood is a measure of 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 the finding were to be exploited by an attacker. 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 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 formally pass to the client;

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.

5 Findings

5.1 [Low] Invalid `batchSize` can result in incorrect hash calculation

File(s): `WorldIDIdentityManagerImplV2.sol`

Description: The function `deleteIdentities(...)` takes a uint32 input `batchSize`, which specifies the number of identities to be deleted. The indices of these identities are packed in the `packedDeletionIndices` input, with each index occupying exactly 32 bits (4 bytes).

```
function deleteIdentities(
    uint256[8] calldata deletionProof,
    uint32 batchSize, // @audit `batchSize` should be the same as the number of deleted elements.
    bytes calldata packedDeletionIndices,
    uint256 preRoot,
    uint256 postRoot
){
    ...
    // @audit batchSize is used to compute `inputHash`
     bytes32 inputHash = calculateIdentityDeletionInputHash(packedDeletionIndices, preRoot, postRoot, batchSize);
    ...
}
```

However, the code lacks a validation check to ensure that the provided `batchSize` aligns with the actual number of elements presented in `packedDeletionIndices`. A mismatch between these two inputs could lead to erroneous `inputHash` calculations within `calculateIdentityDeletionInputHash` function.

```
function calculateIdentityDeletionInputHash(
    bytes calldata packedDeletionIndices,
    uint256 preRoot,
    uint256 postRoot,
    uint32 batchSize
) public view virtual onlyProxy onlyInitialized returns (bytes32 hash) {
    assembly {
        let startOffset := mload(0x40)
        let indicesByteSize := mul(batchSize, 4) // @audit the real size of deletion indices can be different
        calldatacopy(startOffset, packedDeletionIndices.offset, indicesByteSize)
        let rootsOffset := add(startOffset, indicesByteSize)
        mstore(rootsOffset, preRoot)
        mstore(add(rootsOffset, 32), postRoot)
        hash := keccak256(startOffset, add(64, indicesByteSize))
    }
}
```

Recommendation(s): Consider adding a check to ensure that `batchSize` corresponds precisely to the number of elements detailed in `packedDeletionIndices`, i.e., `packedDeletionIndices` should have exactly `(batchSize * 4)` bytes. Furthermore, `batchSize` can potentially be omitted from the inputs, as it can be computed internally using `packedDeletionIndices`.

Status: Fixed

Update from the client: Implemented fix in commit 6821849a and commit 8937c4c3.

5.2 [Info] Possible duplicate roots block propagating root to L2 contracts

File(s): `WorldIDIdentityManagerImplV2.sol`

Description: When an identity is deleted in the identity manager, a new root will be created and then propagated to L2 contracts by state bridge. In the L2 contracts, if a root already exists, it cannot be received again as can be seen in this line. However, it is possible that an existing root is created again after deleting an identity, but since it existed and was progagated before, it will be prevented from propagating to L2 again.

Consider the following scenario:

1. The tree has 4 leaves. The latest root is propagated to L2, containing 4 identities;
2. Two new identities are added, and then two more are added. Now the tree has 8 leaves. The latest root is propagated to L2, containing 8 identities;
3. The last 4 identities are deleted from the tree. The tree now has 4 leaves;

Now, the state of the tree at steps 1 and 3 is the same, making the root after step 3 unable to be propagated. The latest root is still the root of step 2, which contains the deleted identities.

Recommendation(s): Consider reviewing the mechanism for propagating latest root to L2 or preventing duplicated roots in the manager contract to align with the logic in L2.

Status: Acknowledged

Update from the client: In practice we run very big batches (100 or 1000 most of the time) and deletions are highly unlikely to be done from people who just registered. We don't believe that in practice this would be an error that could impact root propagation, since we do hourly batches and a new root would be propagated if for a very odd chance you had the same root for an hour. The signup-sequencer service which handles insertions/deletions among other things also has checks against this happening.

5.3 [Best Practices] Absence of event emission during contract initialization

File(s): `WorldIDIdentityManagerImplV2.sol`

Description: The `initializeV2(...)` function, responsible for initializing the `WorldIDIdentityManagerImplV2` contract, currently lacks an event emission.

```
// @audit Function does not emit an event for the contract initialization
function initializeV2(VerifierLookupTable _batchUpdateVerifiers) public reinitializer(2) {
    batchDeletionVerifiers = _batchUpdateVerifiers;
}
```

Recommendation(s): It is recommended to emit an event upon contract initialization to provide a clear record of this operation and enable better monitoring.

Status: Fixed

Update from the client: Implemented fix in commit 9188a5

5.4 [Best Practices] Floating pragma

File(s): `WorldIDIdentityManagerImplV1.sol` `WorldIDIdentityManagerImplV2.sol`

Description: The Solidity version has been upgraded from `0.8.19` to `0.8.21` in the V2 implementation. However, the contracts still use a floating pragma, which is not recommended and could result in unexpected behavior.

Recommendation(s): Consider locking the version of Solidity to ensure that the contract is deployed with the version it was carefully tested with.

Status: Acknowledged

Update from the client: We have chosen to keep utilizing a floating pragma, since World ID contracts will live only on Ethereum. The solc version used is fixed using foundry.toml.

5.5 [Best Practices] Function `deleteIdentities(...)` has different access control than stated in the documentation

File(s): `WorldIDIdentityManagerImplV2.sol`

Description: The documentation states that only the owner can call the function `deleteIdentities(...)`. However, the function is implemented with the `onlyIdentityOperator` modifier instead of the `onlyOwner` modifier. This means that the function is limited to identity operators instead of the owner.

Recommendation(s): Consider updating the documentation to match the implemented access control or change the modifier to `onlyOwner`.

Status: Fixed

Update from the client: Implemented fix in commit 9eee488.

5.6 [Best Practices] Missing input validation in some functions

File(s): `WorldIDIdentityManagerImplV1.sol` `WorldIDIdentityManagerImplV2.sol`

Description: Certain variables lack validation during initialization and setting operations.

- In the `WorldIDIdentityManagerImplV1` contract, `batchInsertionVerifiers` and `semaphoreVerifier` are not verified to be non-zero addresses during `initialize` and the respective setter functions: `setRegisterIdentitiesVerifierLookupTable` and `setSemaphoreVerifier`;
- In the `WorldIDIdentityManagerImplV2` contract, `batchDeletionVerifiers` is not verified to be a non-zero address on `initialize` and `setDeleteIdentitiesVerifierLookupTable` functions;

Recommendation(s): Consider incorporating input validation checks for these inputs.

Status: Fixed

Update from the client: Fixed in commits 51a6d82b4 and f200c3e

5.7 [Best Practices] Outdated NatSpec documentation and comments in V2 implementation

File(s): `WorldIDIdentityManagerImplV2.sol`

Description: Some NatSpec documentation and comments in `WorldIDIdentityManagerImplV2` contracts seem to be taken from the V1 contract and refer to the insertion logic instead of the deletion.

```
function deleteIdentities(...){
    ...
    try deletionVerifier.verifyProof(deletionProof, [reducedElement]) {
        // If it did verify, we need to update the contract's state. We set the currently valid
        // root to the root after the insertions.
        // @audit old comment referring to insertions.
        _latestRoot = postRoot;
    ...
}
```

```
...
/// @param preRoot The root value of the tree before these insertions were made.
/// @param postRoot The root value of the tree after these insertions were made.
/// @param batchSize The number of identities that were deleted in this batch
// @audit NatSpec documentation referring to insertions
...
function calculateIdentityDeletionInputHash(...)
```

Recommendation(s): Consider updating the documentation and comments to reflect the deletion logic accurately.

Status: Fixed

Update from the client: Implemented fix in commit 9eee488.

5.8 [Best Practices] Used event marked as unused in the documentation

File(s): WorldIDIdentityManagerImplV1.sol

Description: The NatSpec documentation of `TreeChange` structure states that the enum is no longer used in the logic.

```
/// @notice Represents the kind of change that is made to the root of the tree.
/// @dev TreeChange.Update preserved for ABI backwards compatibility with V1, no longer used
// @audit Documentation states that the enum is no longer used.
enum TreeChange {
    Insertion,
    Deletion
}
```

However, the enum is used when emitting `TreeChanged` event in both `registerIdentities` and `deleteIdentities` functions, as shown below.

```
function registerIdentities(...){
    ...
    // @audit `TreeChange` used to emit `TreeChanged` event for insertion
    emit TreeChanged(preRoot, TreeChange.Insertion, postRoot);
    ...
}
```

```
function deleteIdentities(...){
    ...
    // @audit `TreeChange` used to emit `TreeChanged` event for deletion
    emit TreeChanged(preRoot, TreeChange.Deletion, postRoot);
    ...
}
```

Recommendation(s): Update the NatSpec documentation to avoid confusion.

Status: Fixed

Update from the client: Implemented fix in commit a11dfad.

6 Documentation Evaluation

Software documentation refers to the written or visual information describing software's functionality, architecture, design, and implementation. It provides a comprehensive overview of the software system and helps users, developers, and stakeholders understand how the software works, how to use it, and how to maintain it. Software documentation can take different forms, such as user manuals, system manuals, technical specifications, requirements documents, design documents, and code comments. Software documentation is critical in software development, enabling effective communication between developers, testers, users, and other stakeholders. It helps to ensure that everyone involved in the development process has a shared understanding of the software system and its functionality. Moreover, software documentation can improve software maintenance by providing a clear and complete understanding of the software system, making it easier for developers to maintain, modify, and update the software over time. Smart contracts can use various types of software documentation. Some of the most common types include:

- Technical whitepaper: A technical whitepaper is a comprehensive document describing the smart contract's design and technical details. It includes information about the purpose of the contract, its architecture, its components, and how they interact with each other;
- User manual: A user manual is a document that provides information about how to use the smart contract. It includes step-by-step instructions on how to perform various tasks and explains the different features and functionalities of the contract;
- Code documentation: Code documentation is a document that provides details about the code of the smart contract. It includes information about the functions, variables, and classes used in the code, as well as explanations of how they work;
- API documentation: API documentation is a document that provides information about the API (Application Programming Interface) of the smart contract. It includes details about the methods, parameters, and responses that can be used to interact with the contract;
- Testing documentation: Testing documentation is a document that provides information about how the smart contract was tested. It includes details about the test cases that were used, the results of the tests, and any issues that were identified during testing;
- Audit documentation: Audit documentation includes reports, notes, and other materials related to the security audit of the smart contract. This type of documentation is critical in ensuring that the smart contract is secure and free from vulnerabilities.

These types of documentation are essential for smart contract development and maintenance. They help ensure that the contract is properly designed, implemented, and tested, and they provide a reference for developers who need to modify or maintain the contract in the future.

The documentation is presented in README.md. The documentation is sufficient for this audit.

7 Test Suite Evaluation

7.1 Compilation Output

```
> forge compile
[] Compiling...
[] Compiling 94 files with 0.8.21
[] Solc 0.8.21 finished in 36.55s
Compiler run successful with warnings
```

7.2 Tests Output: State Bridge Contracts Upgrade

```
> forge test
[] Compiling...
No files changed, compilation skipped

Running 1 test for src/test/verifier-lookup-table/VerifierLookupTableConstruction.t.sol:VerifierLookupTableConstruction
Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 11.13ms

Running 2 tests for src/test/router/WorldIDRouterDataQuery.t.sol:WorldIDRouterDataQuery
Test result: ok. 2 passed; 0 failed; 0 skipped; finished in 14.66ms

Running 5 tests for
↪  src/test/identity-manager/WorldIDIdentityManagerInitialization.t.sol:WorldIDIdentityManagerInitialization
Test result: ok. 5 passed; 0 failed; 0 skipped; finished in 17.15ms

Running 6 tests for
↪  src/test/verifier-lookup-table/VerifierLookupTableOwnershipManagement.t.sol:VerifierLookupTableOwnershipManagement
Test result: ok. 6 passed; 0 failed; 0 skipped; finished in 258.11ms

Running 16 tests for src/test/identity-manager/WorldIDIdentityManagerUninit.t.sol:WorldIDIdentityManagerUninit
Test result: ok. 16 passed; 0 failed; 0 skipped; finished in 449.28ms

Running 4 tests for src/test/identity-manager/WorldIDIdentityManagerCalculation.t.sol:WorldIDIdentityManagerCalculation
Test result: ok. 4 passed; 0 failed; 0 skipped; finished in 8.33ms

Running 2 tests for
↪  src/test/identity-manager/WorldIDIdentityManagerConstruction.t.sol:WorldIDIdentityManagerConstruction
Test result: ok. 2 passed; 0 failed; 0 skipped; finished in 12.65ms

Running 4 tests for src/test/identity-manager/WorldIDIdentityManagerUpgrade.t.sol:WorldIDIdentityManagerUpdate
Test result: ok. 4 passed; 0 failed; 0 skipped; finished in 548.63ms

Running 2 tests for src/test/router/WorldIDRouterConstruction.t.sol:WorldIDRouterConstruction
Test result: ok. 2 passed; 0 failed; 0 skipped; finished in 217.08ms

Running 9 tests for src/test/verifier-lookup-table/VerifierLookupTableQuery.t.sol.sol:VerifierLookupTableQuery
Test result: ok. 9 passed; 0 failed; 0 skipped; finished in 917.69ms

Running 6 tests for src/test/router/WorldIDRouterOwnershipManagement.t.sol:WorldIDRouterOwnershipManagement
Test result: ok. 6 passed; 0 failed; 0 skipped; finished in 1.01s

Running 5 tests for src/test/router/WorldIDRouterUninit.t.sol:WorldIDRouterUninit
Test result: ok. 5 passed; 0 failed; 0 skipped; finished in 718.21ms

Running 4 tests for src/test/router/WorldIDRouterUpgrade.t.sol:WorldIDRouterUpgrade
Test result: ok. 4 passed; 0 failed; 0 skipped; finished in 512.24ms

Running 11 tests for
↪  src/test/identity-manager/WorldIDIdentityManagerOwnershipManagement.t.sol:WorldIDIdentityManagerOwnershipManagement
Test result: ok. 11 passed; 0 failed; 0 skipped; finished in 2.31s

Running 3 tests for src/test/router/WorldIDRouterStateBridge.t.sol:WorldIDRouterStateBridge
Test result: ok. 3 passed; 0 failed; 0 skipped; finished in 3.75s
```

```
Running 21 tests for
↪  src/test/identity-manager/WorldIDIdentityManagerGettersSetters.t.sol:WorldIDIdentityManagerGettersSetters
Test result: ok. 21 passed; 0 failed; 0 skipped; finished in 3.70s

Running 15 tests for src/test/router/WorldIDRouterRouting.t.sol:WorldIDRouterRouting
Test result: ok. 15 passed; 0 failed; 0 skipped; finished in 4.67s

Running 8 tests for src/test/identity-manager/WorldIDIdentityManagerDataQuery.t.sol:WorldIDIdentityManagerDataQuery
Test result: ok. 8 passed; 0 failed; 0 skipped; finished in 6.90s

Running 3 tests for
↪  src/test/identity-manager/WorldIDIdentityManagerSemaphoreVerification.t.sol:WorldIDIdentityManagerSemaphoreVerification
Test result: ok. 3 passed; 0 failed; 0 skipped; finished in 6.90s

Running 9 tests for
↪  src/test/identity-manager/WorldIDIdentityManagerIdentityDeletion.t.sol:WorldIDIdentityManagerIdentityDeletion
Test result: ok. 9 passed; 0 failed; 0 skipped; finished in 7.11s

Running 12 tests for
↪  src/test/identity-manager/WorldIDIdentityManagerIdentityRegistration.t.sol:WorldIDIdentityManagerIdentityRegistration
Test result: ok. 12 passed; 0 failed; 0 skipped; finished in 8.58s
Ran 21 test suites: 148 tests passed, 0 failed, 0 skipped (148 total tests)
```

7.3 Code Coverage

```
> forge coverage
```

The relevant output is presented below.

```
| File                                        | % Lines          | % Statements      | % Branches      | % Funcs         |
|---------------------------------------------|------------------|-------------------|-----------------|-----------------|
| src/WorldIDIdentityManagerImplV1.sol        | 91.80% (56/61)   | 92.19% (59/64)    | 68.75% (11/16)  | 100.00% (18/18) |
| src/WorldIDIdentityManagerImplV2.sol        | 100.00% (13/13)  | 100.00% (15/15)   | 100.00% (2/2)   | 100.00% (5/5)   |
| src/WorldIDRouterImplV1.sol                 | 96.15% (25/26)   | 96.43% (27/28)    | 100.00% (6/6)   | 90.91% (10/11)  |
| src/abstract/WorldIDImpl.sol                | 100.00% (3/3)    | 100.00% (3/3)     | 100.00% (0/0)   | 100.00% (3/3)   |
| src/data/VerifierLookupTable.sol            | 100.00% (14/14)  | 100.00% (14/14)   | 100.00% (4/4)   | 100.00% (6/6)   |
| src/utils/CheckInitialized.sol              | 100.00% (1/1)    | 100.00% (1/1)     | 100.00% (0/0)   | 100.00% (1/1)   |
| src/utils/SemaphoreTreeDepthValidator.sol | 0.00% (0/3)      | 0.00% (0/3)       | 100.00% (0/0)   | 0.00% (0/1)     |
| src/utils/UnimplementedTreeVerifier.sol   | 0.00% (0/3)      | 0.00% (0/3)       | 100.00% (0/0)   | 0.00% (0/1)     |
```

8 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.