Security Review Report
NM-0131 WORLDCOIN

State Bridge Contracts Upgrade

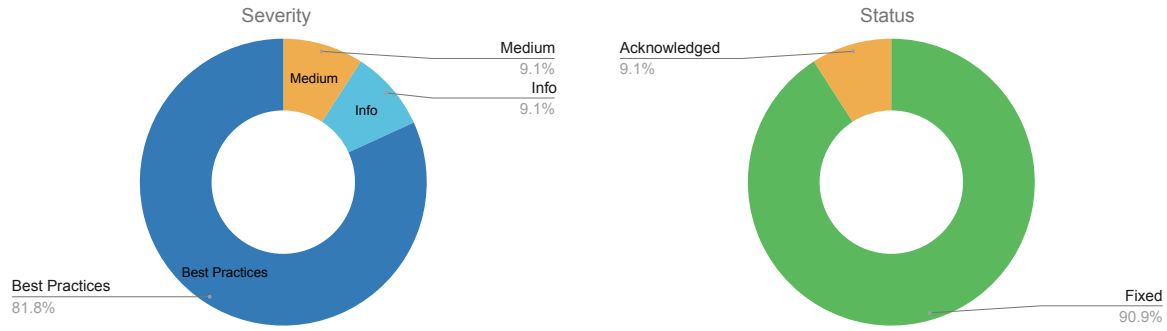
NETHERMIND
SECURITY

(Sep 8, 2023)

Contents

1 Executive Summary

This document presents the security review performed by Nethermind on the State Bridge Contracts Upgrade containing 344 lines of code. The Worldcoin team implemented a significant change in the state bridge mechanism. Instead of pushing the root hash atomically to all chains when inserting identity commitments, they have reorganized the process. Now, distinct tasks handle the data transfer to each supported blockchain independently, without involving the identity manager smart contract, and they do it at various time intervals. **Along the audit of this contract, we report** 11 points of attention, where 1 is classified as `Medium`, 1 is classified as `Info`, and 9 are classified as `Best Practices`. The issues are summarized in Fig. 1.



(a) distribution of issues

(b) status of issues

Fig. 1: Distribution of issues: Critical (0), **High** (0), **Medium** (1), **Low** (0), **Undetermined** (0), **Informational** (1), **Best Practices** (9). **Distribution of status: Fixed** (10), **Acknowledged** (1), **Mitigated** (0), **Unresolved** (0), **Partially Fixed** (0)

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.

Summary of the Audit for the State Bridge Contracts Upgrade

Audit Type	Security Review
Initial Report	Sep. 6, 2023
Response from Client	Sep. 7, 2023
Final Report	Sep. 8, 2023
Methods	Manual Review, Automated Analysis
Repository	world-id-state-bridge
Commit Hash (Audit)	9edc22a94d38870e175c14d0058c4b15bf09bc35
Documentation	README.md
Documentation Assessment	High
Test Suite Assessment	High

2 Audited Files

	Contract	LoC	Comments	Ratio	Blank	Total
1	src/PolygonStateBridge.sol	38	58	152.6%	22	118
2	src/OpStateBridge.sol	82	83	101.2%	38	203
3	src/OpWorldID.sol	14	37	264.3%	8	59
4	src/PolygonWorldID.sol	45	69	153.3%	17	131
5	src/abstract/WorldIDBridge.sol	77	104	135.1%	43	224
6	src/utils/BytesUtils.sol	72	69	95.8%	19	160
7	src/interfaces/IOpStateBridgeTransferOwnership.sol	4	8	200.0%	1	13
8	src/interfaces/IWorldIDIdentityManager.sol	4	5	125.0%	1	10
9	src/interfaces/IPolygonWorldID.sol	4	15	375.0%	2	21
10	src/interfaces/IOpWorldID.sol	4	19	475.0%	2	25
	Total	**344**	**467**	**135.8%**	**153**	**964**

3 Summary of Issues

	Finding	Severity	Update
1	Wrong state variable is set in the function `setGasLimitSendRoot(...)`	Medium	Fixed
2	Transaction ordering in L2 may leave `_latestRoot` in the wrong state after root hash expiration	Info	Acknowledged
3	Avoid using magic numbers	Best Practices	Fixed
4	Inconsistent usage of custom errors	Best Practices	Fixed
5	Incorrect NatSpec description in `_receiveRoot(...)` function	Best Practices	Fixed
6	Lack of event emission when updating state variables	Best Practices	Fixed
7	Missing input validation in some functions	Best Practices	Fixed
8	Optimization of `WorldIdBridge`	Best Practices	Fixed
9	Remove the testing library from the code base	Best Practices	Fixed
10	Remove unused imports	Best Practices	Fixed
11	Typos in the code base	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 [Medium] Wrong state variable is set in the function `setGasLimitSendRoot(...)`

File(s): `OpStateBridge.sol`

Description: The `OpStateBridge` contract allows the owner to customize the gas limit for different methods. The value of `_gasLimitSendRoot` can be set by calling the function `setGasLimitSendRoot(...)`. However, the current codebase instead sets the value for the wrong variable `_gasLimitSetRootHistoryExpiry`.

```
function setGasLimitSendRoot(uint32 _opGasLimit) external onlyOwner {
    // @audit Should set _gasLimitSendRoot instead of _gasLimitSetRootHistoryExpiry
    _gasLimitSetRootHistoryExpiry = _opGasLimit;

    emit SetGasLimitSendRoot(_opGasLimit);
}
```

Recommendation(s): In the function `setGasLimitSendRoot(...)`, set the value for `_gasLimitSendRoot` instead of `_gasLimitSetRootHistoryExpiry`.

Status: Fixed

Update from the client: Fixed, current state.

5.2 [Info] Transaction ordering in L2 may leave `_latestRoot` in the wrong state after root hash expiration

File(s): `WorldIDBridge.sol`

Description: The propagation of `latestRoot` between L1 and L2 is done via messaging. The message data contains the new root hash but does not include the timestamp indicating the emission time from L1. Depending on the different L1-L2 messaging mechanisms, the ordering of transactions cannot be ensured, and it could be possible that two messages sent from L1 to L2 to propagate root get validated in the reverse order on L2.

This scenario has a limited impact since the user on L2 would just need to specify the root hash that contains its identity. However, there is an edge case when the root hash is not updated and expiration time (`ROOT_HISTORY_EXPIRY`) is reached for all root hashes. The only valid root hash is the latest one (`_latestRoot`), which could be the second to last one sent from L1.

```
function requireValidRoot(uint256 root) internal view {
    // The latest root is always valid.
    if (root == _latestRoot) {
        return;
    }
```

All users registered between these two root hash propagations cannot create valid proofs on L2 until a new root hash is propagated.

Recommendation(s): Consider sending the timestamp from L1 to ensure the latest root hash is correctly updated.

Status: Acknowledged

Update from the client: Acknowledged. Transaction ordering doesn't impact the correct functioning of bridged World IDs. The `signup-sequencer` is the one that takes care of providing roots against the latest root on the L2, which solves the edge case mentioned above.

5.3 [Best Practices] Avoid using magic numbers

File(s): OpStateBridge.sol, PolygonWorldID.sol

Description: Enhancing code readability can be achieved by replacing magic numbers with meaningful constants. The code snippet below shows an example:

```
constructor(
    address _worldIDIdentityManager,
    address _opWorldIDAddress,
    address _crossDomainMessenger
) {
    opWorldIDAddress = _opWorldIDAddress;
    worldIDAddress = _worldIDIdentityManager;
    crossDomainMessengerAddress = _crossDomainMessenger;
    // @audit Avoid using magic number
    _gasLimitSendRoot = 100000;
    _gasLimitSetRootHistoryExpiry = 100000;
    _gasLimitTransferOwnership = 100000;
}
```

Recommendation(s): Consider replacing these magic numbers with named constants or variables that provide meaningful names and context for these values.

Status: Fixed

Update from the client: Fixed in commits 364826209, 33df6b53

5.4 [Best Practices] Inconsistent usage of custom errors

File(s): PolygonStateBridge.sol, PolygonWorldID.sol

Description: Within these contract, both `require()/revert()` and custom errors are employed. Adopting a consistent approach and using only one error-handling method within each file is recommended.

Recommendation(s): Consider adopting a uniform error-handling method throughout the codebase.

Status: Fixed

Update from the client: Fixed in commit 52f2c77

5.5 [Best Practices] Incorrect NatSpec description in `_receiveRoot(...)` function

File(s): WorldIDBridge.sol

Description: The `_receiveRoot(...)` function contains incorrect custom NatSpec. It specifies that the function may revert if the caller is not the owner. However, the function does not have this logic.

```
/// @custom: reverts string If the caller is not the owner.
function _receiveRoot(uint256 newRoot) internal {
[...]
}
```

Recommendation(s): Consider correcting the NatSpec documentation to reflect the function's implementation.

Status: Fixed

Update from the client: Fixed in commit 844da90

5.6 [Best Practices] Lack of event emission when updating state variables

File(s): PolygonStateBridge.sol, PolygonWorldID.sol

Description: It is best practice to emit an event when changing important state variables. The code snippet below shows an example where the function sets the address for `fxRootTunnel`, but does not emit an event.

```
function setFxRootTunnel(address _fxRootTunnel) external virtual override onlyOwner {
    require(fxRootTunnel == address(0x0), "FxBaseChildTunnel: ROOT_TUNNEL_ALREADY_SET");
    // @audit Should emit event
    fxRootTunnel = _fxRootTunnel;
}
```

Recommendation(s): Consider emitting a suitable event when updating state variables.

Status: Fixed

Update from the client: Fixed in commits 33df6b53, c3c25679

5.7 [Best Practices] Missing input validation in some functions

File(s): OpStateBridge.sol, PolygonStateBridge.sol, PolygonWorldID.sol

Description: Some functions do not check for the validity of their input before utilizing them:

- `OpStateBridge.transferOwnershipOp(...)` does not check if `_owner` is different than `address(0)`. Note that a check is implemented in `transferOwnership` function of `CrossDomainOwnable3` contract, but it can be endorsed at the protocol contract level;
- gas limit setters (`setGasLimitSendRoot(...)`, `setGasLimitSetRootHistoryExpiry(...)`, `setGasLimitTransferOwnershipOp(...)` ..) does not check for the validity of the gas limit (i.e., whether it differs from 0);
- `PolygonStateBridge` constructor function lacks validation for inputs `_checkpointManager` `_fxRoot` and `_worldIDIdentityManager`, allowing them to be set as `address(0)`;
- `PolygonWorldID` contract constructor lacks validation of `_fxChild` address;

Including input validation to enhance code robustness and prevent unexpected errors is a good practice.

Recommendation(s): We recommend implementing a check for inputs in different functions.

Status: Fixed

Update from the client: Fixed in commits c3c25679, 0c08228

5.8 [Best Practices] Optimization of `WorldIdBridge`

File(s): WorldIDBridge.sol

Description: The `treeDepth` state variable from the `WorldIDBridge` abstract contract is never updated. This value is only set within the `constructor`.

```
/// @notice Constructs a new instance of the state bridge.
///
/// @param _treeDepth The depth of the identities Merkle tree.
constructor(uint8 _treeDepth) {
    if (!SemaphoreTreeDepthValidator.validate(_treeDepth)) {
        revert UnsupportedTreeDepth(_treeDepth);
    }

    treeDepth = _treeDepth;
}
```

Recommendation(s): Gas optimization can be done following these steps:

- change `treedepth` to an immutable variable (initialization in constructor);
- change type of `getTreeDepth()` function to `pure`;

Status: Fixed

Update from the client: `getTreeDepth()` needs to remain a `view` function for the contract to compile. The solidity compiler considers immutable variables to be non-pure because it is possible to initialize them with some state link. Fixed in commits 12d09df, 8e0e5b4

5.9 [Best Practices] Remove the testing library from the code base

File(s): WorldIDBridge.sol

Description: The `WorldIdBridge` abstract contract imports the `forge-std` console library.

```
import "forge-std/console.sol";
```

Recommendation(s): Consider removing the testing library in the production code base.

Status: Fixed

Update from the client: Fixed in commit 88beb2e

5.10 [Best Practices] Remove unused imports

File(s): PolygonWorldID.sol

Description: `PolygonWorldID` contract has the following imports that are not being used.

```
import {SemaphoreTreeDepthValidator} from "./utils/SemaphoreTreeDepthValidator.sol";
import {SemaphoreVerifier} from "semaphore/base/SemaphoreVerifier.sol";
```

Recommendation(s): Consider removing unused imports from the contract.

Status: Fixed

Update from the client: Fixed in commit 33df6b5

5.11 [Best Practices] Typos in the code base

File(s): WorldIDBridge.sol, OpStateBridge.sol

Description: The code base contains the following typos:

- The NatSpec documentation from the `verifyProof(...)` function of the `WorldIDBridge` contract;

```
/// @param root The of the Merkle tree
```

- The NatSpec documentation from the `OpStateBridge` contract:;

```
/// @notice Emitted when the the StateBridge sets the gas limit for SetRootHistoryExpiryt
/// @param _opGasLimit The new opGasLimit for SetRootHistoryExpirytimism
```

Recommendation(s): Consider correcting the typos to improve code quality.

Status: Fixed

Update from the client: Fixed in commits d89e337, 399dec2, e986574

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
- skipping the remapping
[] Compiling...
[] Compiling 152 files with 0.8.15
[] Solc 0.8.15 finished in 12.05s
Compiler run successful with warnings:
Warning (6321): Unnamed return variable can remain unassigned. Add an explicit return with value to all non-reverting
 ↪  code paths or name the variable.
  --> lib/contracts/contracts/lib/MerklePatriciaProof.sol:20:30:
   |
20 |     ) internal pure returns (bool) {
   |
```

7.2 Tests Output: State Bridge Contracts Upgrade

```
> forge test
- skipping the remapping
[] Compiling...
No files changed, compilation skipped

Running 4 tests for src/test/MockPolygonBridge.t.sol:MockPolygonBridgeTest
[PASS] test_ReceiveRoot_succeeds(uint256) (runs: 256, : 62076, ~: 62776)
[PASS] test_processMessageFromRoot_reverts_CannotOverwriteRoot(uint256) (runs: 256, : 67733, ~: 67733)
[PASS] test_processMessageFromRoot_reverts_InvalidMessageSelector(bytes4,bytes32) (runs: 256, : 16496, ~: 16496)
[PASS] test_setRootHistoryExpiry_succeeds(uint256) (runs: 256, : 22737, ~: 22850)
Test result: ok. 4 passed; 0 failed; 0 skipped; finished in 34.91ms

Running 5 tests for src/test/BytesUtils.t.sol:BytesUtilsTest
[PASS] testDifferentSigsDontCollideSucceeds(string,string) (runs: 256, : 5095, ~: 5055)
[PASS] testGrabSelectorPayloadTooShortReverts(bytes2,bytes4) (runs: 256, : 3510, ~: 3510)
[PASS] testGrabSelectorSucceeds(string) (runs: 256, : 1707, ~: 1730)
[PASS] testStripSelectorPayloadTooShortReverts(bytes4,bytes5) (runs: 256, : 3788, ~: 3788)
[PASS] testStripSelectorSucceeds(string) (runs: 256, : 1950, ~: 1967)
Test result: ok. 5 passed; 0 failed; 0 skipped; finished in 72.19ms

Running 6 tests for src/test/PolygonWorldID.t.sol:PolygonWorldIDTest
[PASS] testCanGetTreeDepth(uint8) (runs: 256, : 7253824, ~: 7253824)
[PASS] testConstructorWithInvalidTreeDepth(uint8) (runs: 256, : 6333808, ~: 6333833)
[PASS] testSetRootHistoryExpiryReverts(uint256) (runs: 256, : 8562, ~: 8562)
[PASS] test_notOwner_acceptOwnership_reverts(address,address) (runs: 256, : 38730, ~: 38730)
[PASS] test_owner_renounceOwnership_reverts() (gas: 12848)
[PASS] test_owner_transferOwnership_succeeds(address) (runs: 256, : 30448, ~: 30526)
Test result: ok. 6 passed; 0 failed; 0 skipped; finished in 190.25ms

Running 8 tests for src/test/OpWorldID.t.sol:OpWorldIDTest
[PASS] testCanGetTreeDepth(uint8) (runs: 256, : 7017319, ~: 7017319)
[PASS] testConstructorWithInvalidTreeDepth(uint8) (runs: 256, : 6353475, ~: 6353498)
[PASS] test_expiredRoot_reverts(uint256,uint256,uint256[8]) (runs: 256, : 180360, ~: 180360)
[PASS] test_onlyOwner_notMessenger_reverts(uint256) (runs: 256, : 28545, ~: 28545)
[PASS] test_onlyOwner_notOwner_reverts(uint256) (runs: 256, : 32248, ~: 32248)
[PASS] test_receiveRoot_reverts_CannotOverwriteRoot(uint256) (runs: 256, : 155673, ~: 155688)
[PASS] test_receiveVerifyInvalidRoot_reverts(uint256,uint256[8]) (runs: 256, : 119485, ~: 120263)
[PASS] test_receiveVerifyRoot_succeeds(uint256) (runs: 256, : 116371, ~: 116371)
Test result: ok. 8 passed; 0 failed; 0 skipped; finished in 208.60ms

Running 13 tests for src/test/OpStateBridge.t.sol:OpStateBridgeTest
[PASS] test_canSelectFork_succeeds() (gas: 5696)
[PASS] test_notOwner_SetRootHistoryExpiry_reverts(address,uint256) (runs: 256, : 13727, ~: 13727)
[PASS] test_notOwner_acceptOwnership_reverts(address,address) (runs: 256, : 21774, ~: 21774)
[PASS] test_notOwner_transferOwnershipOp_reverts(address,address,bool) (runs: 256, : 13976, ~: 13976)
```

```
[PASS] test_notOwner_transferOwnership_reverts(address,address) (runs: 256, : 13929, ~: 13929)
[PASS] test_owner_renounceOwnership_reverts() (gas: 12890)
[PASS] test_owner_setGasLimitSendRoot_succeeds(uint32) (runs: 256, : 20656, ~: 20656)
[PASS] test_owner_setGasLimitSetRootHistoryExpiry_succeeds(uint32) (runs: 256, : 20679, ~: 20679)
[PASS] test_owner_setGasLimitTransferOwnershipOp_succeeds(uint32) (runs: 256, : 20690, ~: 20690)
[PASS] test_owner_setRootHistoryExpiry_succeeds(uint256) (runs: 256, : 84218, ~: 84218)
[PASS] test_owner_transferOwnershipOp_succeeds(address,bool) (runs: 256, : 87551, ~: 87551)
[PASS] test_owner_transferOwnership_succeeds(address) (runs: 256, : 28167, ~: 28167)
[PASS] test_propagateRoot_suceeds() (gas: 86640)
Test result: ok. 13 passed; 0 failed; 0 skipped; finished in 120.56s

Running 8 tests for src/test/PolygonStateBridge.t.sol:PolygonStateBridgeTest
[PASS] test_canSelectFork_succeeds() (gas: 5651)
[PASS] test_notOwner_acceptOwnership_reverts(address,address) (runs: 256, : 38900, ~: 38900)
[PASS] test_notOwner_setRootHistoryExpiryPolygon_reverts(address,uint256) (runs: 256, : 13663, ~: 13663)
[PASS] test_notOwner_transferOwnership_reverts(address,address) (runs: 256, : 13863, ~: 13863)
[PASS] test_owner_renounceOwnership_reverts() (gas: 12905)
[PASS] test_owner_setRootHistoryExpiryPolygon_succeeds(uint256) (runs: 256, : 43348, ~: 43348)
[PASS] test_owner_transferOwnership_succeeds(address) (runs: 256, : 34360, ~: 34344)
[PASS] test_propagateRoot_succeeds(address) (runs: 256, : 48336, ~: 48336)
Test result: ok. 8 passed; 0 failed; 0 skipped; finished in 120.65s
Ran 6 test suites: 44 tests passed, 0 failed, 0 skipped (44 total tests)
```

7.3 Code Coverage

```
> forge coverage
```

The relevant output is presented below.

```
Analysing contracts...
Running tests...
| src/OpStateBridge.sol          | 100.00% (19/19)   | 100.00% (21/21)  | 100.00% (0/0)  | 100.00% (7/7)   |
| src/OpWorldID.sol              | 50.00% (1/2)      | 50.00% (1/2)     | 100.00% (0/0)  | 50.00% (1/2)    |
| src/PolygonStateBridge.sol     | 81.82% (9/11)     | 84.62% (11/13)   | 0.00% (0/2)    | 60.00% (3/5)    |
| src/PolygonWorldID.sol         | 15.38% (2/13)     | 11.76% (2/17)    | 0.00% (0/6)    | 50.00% (2/4)    |
| src/abstract/WorldIDBridge.sol | 86.96% (20/23)    | 88.00% (22/25)   | 70.00% (7/10)  | 85.71% (6/7)    |
| src/utils/BytesUtils.sol       | 63.16% (12/19)    | 68.18% (15/22)   | 16.67% (1/6)   | 50.00% (2/4)    |
```

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.