Security Review Report
NM-0777 - Immutable

NETHERMIND SECURITY

(Febuary 9, 2026)

Contents

1 Executive Summary

This document presents the results of a security review conducted by Nethermind Security for Immutable's **Asset Migration Contracts**. This system facilitates the transition of user funds from the legacy StarkEx bridge to Immutable zkEVM following the sunset of the Immutable X chain. The architecture enables the phased migration of remaining ETH and ERC-20 holdings while preserving the ability for users to finalize pre-sunset pending withdrawals.

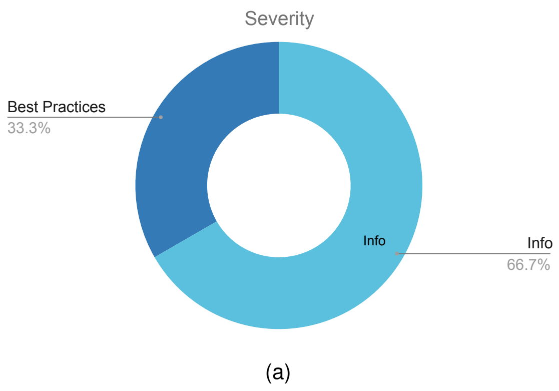
The migration process utilizes Axelar for cross-chain state propagation, anchoring the system on the final cryptographic vault root to ensure assets are disbursed only to their rightful owners. On-chain security is maintained through a disbursement engine on zkEVM that verifies vault ownership and account associations via Pedersen and Keccak256 Merkle proof validation. While the system employs permissioned operators for operational efficiency, all fund releases are strictly constrained by the authenticated state committed prior to the legacy bridge upgrade. This design ensures a trust-minimized process where assets are mapped to their corresponding zkEVM addresses and protected against double-disbursement.

The audit comprises 865 lines of the Solidity code. **The audit was performed using** (a) manual analysis of the codebase, and (b) automated analysis tools.

Along this document, we report 3 points of attention, where two are classified as `Informational` and one is classified as `Best Practices`.

The issues are summarized in Fig. 1.

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



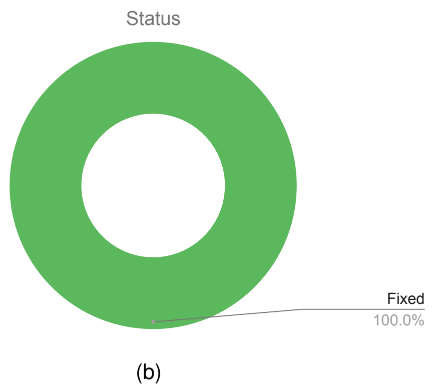


Fig. 1: Distribution of issues: Critical (0), **High** (0), **Medium** (0), **Low** (0), **Undetermined** (0), **Informational** (2), **Best Practices** (1).
Distribution of status: Fixed (3), **Acknowledged** (0), **Mitigated** (0), **Unresolved** (0)

Summary of the Audit

Audit Type	Security Review
Initial Report	February 06, 2026
Final Report	February 09, 2026
Initial Commit	`9a20a518245ad9517d2e9f6a6398158e331b5701`
Final Commit	`358732189061b4887feafb7c99c161a1bce3ff95`
Documentation Assessment	High
Test Suite Assessment	High

2 Audited Files

	Contract	LoC	Comments	Ratio	Blank	Total
1	src/bridge/starkex/IStarkExchangeMigration.sol	21	35	166.7%	12	68
2	src/bridge/starkex/ProxyStorage.sol	7	20	285.7%	5	32
3	src/bridge/starkex/StarkExchangeMigration.sol	74	44	59.5%	17	135
4	src/bridge/starkex/MainStorage.sol	44	57	129.5%	37	138
5	src/bridge/starkex/LegacyStarkExchangeBridge.sol	118	130	110.2%	31	279
6	src/bridge/starkex/GovernanceStorage.sol	9	19	211.1%	3	31
7	src/bridge/starkex/libraries/Common.sol	38	33	86.8%	7	78
8	src/bridge/zkEVM/IRootERC20Bridge.sol	8	38	475.0%	4	50
9	src/bridge/messaging/VaultRootSenderAdapter.sol	49	39	79.6%	20	108
10	src/bridge/messaging/VaultRootReceiverAdapter.sol	59	53	89.8%	25	137
11	src/verifiers/vaults/VaultEscapeProofVerifier.sol	155	153	98.7%	50	358
12	src/verifiers/vaults/IVaultProofVerifier.sol	16	46	287.5%	6	68
13	src/verifiers/accounts/AccountProofVerifier.sol	14	24	171.4%	3	41
14	src/withdrawals/ProcessorAccessControl.sol	46	17	37.0%	8	71
15	src/withdrawals/VaultWithdrawalProcessor.sol	107	68	63.6%	33	208
16	src/withdrawals/VaultRootReceiver.sol	14	27	192.9%	6	47
17	src/withdrawals/BaseVaultWithdrawalProcessor.sol	24	49	204.2%	8	81
18	src/withdrawals/AccountRootReceiver.sol	16	7	43.8%	9	32
19	src/assets/BridgedTokenMapping.sol	46	75	163.0%	19	140
	Total	**865**	**934**	**108.0%**	**303**	**2102**

3 Summary of Issues

	Finding	Severity	Update
1	Inconsistent vault proof length validation restricts tree height	Info	Fixed
2	Role-based access for vault root setting reduces trust minimization	Info	Fixed
3	The implementation contract is not initialized	Best Practices	Fixed

4 System Overview

The Immutable X migration system is a specialized suite of smart contracts designed to facilitate the secure transfer of user assets from the legacy StarkEx-based Immutable X platform to the Immutable zkEVM environment. As the Immutable X chain is scheduled for sunset in early 2026, this system provides an automated mechanism for users who did not withdraw their funds manually prior to the shutdown. To ensure the process is trust-minimized, the architecture anchors on the final cryptographic state (the vault root) of the legacy bridge and enforces on-chain verification for all disbursements on the destination chain. While specific roles like the `Disburser` and `Migration Initiator` are permissioned for operational efficiency, they are strictly gated by cryptographic proofs, preventing the unauthorized diversion of funds.

4.1 State Finalization and Root Propagation

The migration process begins with the commitment of the final `Vault Root` and `Account Root` on Ethereum (L1) and their subsequent propagation to Immutable zkEVM (L2).

- **Vault Root Commitment**: The `StarkExchangeMigration` contract on L1 acts as the final upgraded implementation of the legacy bridge, preserving the final state root representing all user vault balances at the time of sunset.
- **Account Root Commitment**: Because there is no comprehensive on-chain mapping of Stark Keys to Ethereum addresses, an `Account Mapping` is generated off-chain and its Merkle root is committed to the `VaultWithdrawalProcessor` on zkEVM. To mitigate trust, these mappings and the corresponding root are published for public review before the system is finalized.
- **Cross-Chain Messaging**: The `VaultRootSenderAdapter` on L1 utilizes the Axelar Gateway to transmit the final vault root to the `VaultRootReceiverAdapter` on L2. This ensures that the destination chain has an authenticated "source of truth" to verify against.
- **Immutability and Timelocks**: The upgrade to the L1 bridge is subject to a 14-day mandatory timelock. On L2, administrative controls and root override capabilities are intended to be irrevocably renounced once the system is fully initialized, ensuring the migration parameters cannot be altered after the fact.

4.2 Asset Migration (L1 to L2)

Once the state is finalized, the actual holdings (ETH and ERC-20 tokens) must be moved from the legacy L1 bridge to the zkEVM bridge infrastructure.

- **Phased Migration**: A `migrationManager` initiates the transfer of assets in administrator-defined, bounded phases. This phased approach manages liquidity and reduces the impact of any potential operational issues.
- **Bridge Integration**: The `StarkExchangeMigration` contract interacts with the zkEVM bridge to deposit funds. These funds are directed specifically to the `VaultWithdrawalProcessor` on zkEVM, which acts as the recipient for migrated assets.
- **Legacy Finalization**: During this phase, the system still enables users to finalize "pending withdrawals" that were initiated on the legacy bridge before the sunset, ensuring no pre-existing transactions are left stranded.

4.3 Proof-Gated Disbursement

The final stage is the disbursement of funds on Immutable zkEVM to the rightful owners, handled by the `VaultWithdrawalProcessor`.

- **Vault and Account Verification**: The disbursement process is driven by a permissioned entity that must provide a valid `Vault Proof` and `Account Proof` for each migration. The `VaultEscapeProofVerifier` confirms the vault balance against the final root using Pedersen hash-based Merkle proof validation. Simultaneously, the `AccountProofVerifier` validates the association between a Stark Key and an Ethereum address using a standard Keccak256 Merkle proof mechanism leveraging OpenZeppelin's `MerkleProof` library.
- **Asset Mapping**: The `BridgedTokenMapping` contract translates legacy Immutable X asset IDs and quantum into modern zkEVM token addresses. This ensures the processor distributes the correct equivalent asset on the new chain.
- **Double-Spending Prevention**: The system maintains a `processedWithdrawals` registry. Every successful claim is recorded based on the user's Stark Key and asset ID, ensuring that each vault can only be disbursed exactly once across all migration phases.
- **Operator Accountability**: The disbursement function is executed by a permissioned `Disburser`. While this operator is trusted to execute the process in a timely manner to prevent funds from being stranded, they cannot manipulate recipients or amounts, as every transaction remains strictly gated by on-chain cryptographic verification.

5 Risk Rating Methodology

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

Likelihood measures how likely the finding is to be uncovered and exploited by an attacker. This factor will be one of the following values:

a) **High**: The issue is trivial to exploit and has no specific conditions that need to be met;

b) **Medium**: The issue is moderately complex and may have some conditions that need to be met;

c) **Low**: The issue is very complex and requires very specific conditions to be met.

When defining the likelihood of a finding, other factors are also considered. These can include but are not limited to motive, opportunity, exploit accessibility, ease of discovery, and ease of exploit.

Impact is a measure of the damage that may be caused if an attacker exploits the finding. This factor will be one of the following values:

a) **High**: The issue can cause significant damage, such as loss of funds or the protocol entering an unrecoverable state;

b) **Medium**: The issue can cause moderate damage, such as impacts that only affect a small group of users or only a particular part of the protocol;

c) **Low**: The issue can cause little to no damage, such as bugs that are easily recoverable or cause unexpected interactions that cause minor inconveniences.

When defining the impact of a finding, other factors are also considered. These can include but are not limited to Data/state integrity, loss of availability, financial loss, and reputation damage. After defining the likelihood and impact of an issue, the severity can be determined according to the table below.

		Severity Risk		
Impact	**High**	Medium	High	Critical
	Medium	Low	Medium	High
	Low	Info/Best Practices	Low	Medium
	Undetermined	Undetermined	Undetermined	Undetermined
		Low	**Medium**	**High**
		Likelihood		

To address issues that do not fit a High/Medium/Low severity, Nethermind Security also uses three more finding severities: **Informational**, **Best Practices**, and **Undetermined**.

a) **Informational** findings do not pose any risk to the application, but they carry some information that the audit team intends to pass to the client formally;

b) **Best Practice** findings are used when some piece of code does not conform with smart contract development best practices;

c) **Undetermined** findings are used when we cannot predict the impact or likelihood of the issue.

6 Issues

6.1 [Info] Inconsistent vault proof length validation restricts tree height

File(s): src/withdrawals/VaultWithdrawalProcessor.sol

Description: The VaultWithdrawalProcessor contract enforces a strict equality check on the vault proof length, limiting it to exactly 68 words (Tree Height 31).

```
function verifyAndProcessWithdrawal(...) {
    // ...
    // @audit strict check on vault proof length
    require(
        vaultProof.length == VAULT_PROOF_LENGTH,
        IVaultProofVerifier.InvalidVaultProof("Invalid vault proof length")
    );
    // ...
}
```

This contradicts the logic in the underlying VaultEscapeProofVerifier, which is explicitly designed to support variable-length proofs for Merkle tree heights ranging from 31 to 97 (proof lengths of 68 to 199 words).

```
function _validateProofStructure(uint256[] calldata escapeProof) private pure returns (bool) {
    // ...
    // @audit flexible check on vault proof length
    require(proofLength >= VAULT_PROOF_LENGTH, InvalidVaultProof("Proof too short."));
    require(proofLength < 200, InvalidVaultProof("Proof too long."));
    // ...
}
```

By enforcing strict equality in the calling contract, the VaultWithdrawalProcessor effectively disables the verifier's capability to handle larger Merkle trees.

Technically, this hardcoded check creates a denial-of-service vector for any valid proof corresponding to a tree height greater than 31. If the L2 system were to upgrade its tree height to accommodate more users, the VaultWithdrawalProcessor would reject those valid proofs, permanently trapping funds.

However, since this contract is deployed specifically for a one-time migration and is expected to be deprecated in the near future, the likelihood of a Merkle tree height upgrade occurring during its short operational lifespan is negligible. Therefore, while the logic is inconsistent, the practical risk to the migration process is minimal.

Recommendation(s): Consider removing the redundant vault proof length check, ensuring the remaining check reflects the expected proof length.

Status: Fixed

Update from the client: The Immutable X Vault Merkle tree depth is fixed at 31, consistent with other StarkEx Spot trading chains. For a tree of this depth, the expected vault proof length is exactly 68 words, which informs the validation enforced in VaultWithdrawalProcessor.sol. As a result, this check does not introduce a denial-of-service vector as alluded to, but instead correctly enforces the only valid proof structure for Immutable X. The more permissive validation in VaultEscapeProofVerifier, inherited from the original StarkEx implementation, supports broader vault tree structures, beyond what is applicable to Immutable X. However, we acknowledge that this discrepancy introduces ambiguity.

To address this, we have aligned the VaultEscapeProofVerifier's validation with the stricter, fixed-length requirement applicable to Immutable X. Additionally, to avoid duplication in validation in the standard withdrawal path, the vault proof length check has been removed from VaultWithdrawalProcessor, with validation now performed exclusively in VaultEscapeProofVerifier.

These changes were applied as part of PR #8, on commit cd7424c

6.2 [Info] Role-based access for vault root setting reduces trust minimization

File(s): `src/withdrawals/VaultWithdrawalProcessor.sol`

Description: The system is designed to propagate the vault root from Ethereum to zkEVM via a trust-minimized cross-chain flow (StarkEx → Axelar → `VaultRootReceiverAdapter` → `VaultWithdrawalProcessor`). This architecture is intended to ensure that the vault root stored on zkEVM is cryptographically authenticated and originates specifically from the legitimate L1 bridge.

However, the `setVaultRoot()` function in `VaultWithdrawalProcessor` relies on a generic role-based access control (`VAULT_ROOT_PROVIDER_ROLE`) rather than requiring the call to originate specifically from the authenticated bridge adapter.

```
function setVaultRoot(uint256 newRoot) external override onlyRole(VAULT_ROOT_PROVIDER_ROLE) {
    _setVaultRoot(newRoot, rootOverrideAllowed);
}
```

This design allows any address granted this role (e.g., an EOA or Multisig) to manually inject a vault root, bypassing the Axelar validation entirely. This undermines the trust model of the vault root propagation mechanism and undermines trust minimization, a core principle the system aims to achieve.

Recommendation(s): Consider restricting access to `setVaultRoot` so that it can only be called by the `VaultRootReceiverAdapter`, which can be defined as an immutable variable in the contract.

Status: Fixed

Update from the client: The generic role-based mechanism was introduced to provide operational flexibility during testing and pre-migration phases. As with other privileged roles in the contract, the intended operational model is that all such roles are fully renounced before the migration upgrade contracts are proposed on-chain and enter the timelock period, enabling users to fully audit the final security parameters of these contracts.

However, we agree that this implementation does not make the intended trust model sufficiently explicit and leaves room for operational deviation. To make this intent explicit and enforceable at the contract level, we have implemented the recommended change.

These changes were applied as part of PR #8, on commit 7b63b20

6.3 [Best Practices] The implementation contract is not initialized

File(s): `src/bridge/starkex/StarkExchangeMigration.sol`

Description: The `StarkExchangeMigration` contract uses the `Initializable` pattern from OpenZeppelin to support its deployment behind an upgradeable proxy. It includes an `initialize(...)` function to set protocol addresses such as the `migrationManager`, `zkEVMBridge`, and `rootSenderAdapter`.

However, the contract does not include a constructor that calls `_disableInitializers()`. When using upgradeable contracts, it is a standard security practice to disable initializers on the implementation contract itself. Without this, the implementation contract remains uninitialized in its own storage context, allowing any user to call the `initialize(...)` function directly on the implementation address.

While this does not directly affect the security or operation of the proxy since it maintains its own separate storage, it is considered a best practice to lock the implementation. Leaving it uninitialized can lead to user confusion when inspecting the contract state on-chain, as the storage variables on the implementation address could be set to misleading values.

Recommendation(s): Consider adding a constructor to the `StarkExchangeMigration` contract that calls `_disableInitializers()` to ensure the implementation contract cannot be initialized.

Status: Fixed

Update from the client: We agree with the recommendation.

These changes were applied as part of PR #8, on commit 55643ed

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 Immutable's documentation

The `Immutable asset migration contracts` documentation was primarily delivered via a comprehensive `README` file, which offered a detailed architectural walkthrough of the migration process, its distinct phases, and the core components involved. This was further complemented by thorough NatSpec comments within the codebase, clearly defining the purpose and expected behavior of each function.

In addition to the written materials, the client provided a technical deep dive and migration plan overview during the initial kickoff call. Throughout the engagement, the `Immutable` team addressed all questions and concerns raised by the `Nethermind Security` team during their regular meetings.

8 Test Suite Evaluation

8.1 Tests Output

```
> forge test

Ran 4 tests for test/proof-of-code/ProofOfFrontRunInitialize.sol:ProofOfFrontRunInitialize
[PASS] test_AnyoneCanCallInitialize() (gas: 158247)
[PASS] test_AttackerControlsPrivilegedFunctions() (gas: 169838)
[PASS] test_CompleteAttackScenario() (gas: 184438)
[PASS] test_LegitOwnerCannotReinitialize() (gas: 160375)
Suite result: ok. 4 passed; 0 failed; 0 skipped; finished in 8.10ms (2.59ms CPU time)

Ran 12 tests for test/unit/bridge/messaging/VaultRootSenderAdapter.t.sol:VaultRootSenderAdapterTest
[PASS] test_Constructor() (gas: 42855)
[PASS] test_Execute_NotSupported() (gas: 19730)
[PASS] test_RevertIf_Constructor_InvalidVaultReceiverAddress() (gas: 47033)
[PASS] test_RevertIf_Constructor_InvalidVaultReceiverChainId() (gas: 46670)
[PASS] test_RevertIf_Constructor_ZeroGasServiceAddress() (gas: 46348)
[PASS] test_RevertIf_Constructor_ZeroGatewayAddress() (gas: 45103)
[PASS] test_RevertIf_Constructor_ZeroRootSenderAddress() (gas: 48042)
[PASS] test_RevertIf_SendVaultRoot_CallerNotBridge() (gas: 10899)
[PASS] test_RevertIf_SendVaultRoot_InvalidVaultRoot() (gas: 20781)
[PASS] test_RevertIf_SendVaultRoot_NoBridgeFee() (gas: 13062)
[PASS] test_RevertIf_SendVaultRoot_ZeroGasRefundAddress() (gas: 21166)
[PASS] test_SendVaultRoot() (gas: 71691)
Suite result: ok. 12 passed; 0 failed; 0 skipped; finished in 8.13ms (2.54ms CPU time)

Ran 11 tests for test/unit/verifiers/accounts/AccountProofVerifier.t.sol:AccountProofVerifierTest
[PASS] test_ComputeLeafHash_Consistency() (gas: 28032)
[PASS] test_ErrorSelector() (gas: 8824)
[PASS] test_RevertIf_InvalidAccountRoot() (gas: 15181)
[PASS] test_RevertIf_InvalidEthAddress() (gas: 17692)
[PASS] test_RevertIf_InvalidMerkleProof() (gas: 16595)
[PASS] test_RevertIf_InvalidStarkKey() (gas: 17777)
[PASS] test_VerifyAccountProof_EdgeCases() (gas: 28055)
[PASS] test_VerifyAccountProof_MultipleProofs() (gas: 51022)
[PASS] test_VerifyAccountProof_WithLongProof() (gas: 23393)
[PASS] test_VerifyValidAccountProof() (gas: 18047)
[PASS] test_VerifyValidAccountProof_WithMerklePath() (gas: 20288)
Suite result: ok. 11 passed; 0 failed; 0 skipped; finished in 8.13ms (2.54ms CPU time)

Ran 14 tests for test/unit/withdrawals/AccountRootReceiver.t.sol:AccountRootReceiverTest
[PASS] test_AccountRootSet_EventEmitted() (gas: 39623)
[PASS] test_AccountRootSet_EventEmitted_Override() (gas: 45058)
[PASS] test_AccountRootValidation() (gas: 15330)
[PASS] test_EdgeCases() (gas: 85560)
[PASS] test_ErrorSelectors() (gas: 19939)
[PASS] test_EventParameters() (gas: 39777)
[PASS] test_InitialState() (gas: 25002)
[PASS] test_MultipleRoots_WithOverride() (gas: 79672)
[PASS] test_OverrideSetting_Change() (gas: 74913)
[PASS] test_RevertIf_SetAccountRoot_OverrideNotAllowed() (gas: 44637)
[PASS] test_RevertIf_SetAccountRoot_ZeroValue() (gas: 13249)
[PASS] test_SetAccountRoot_Initial() (gas: 43958)
[PASS] test_SetAccountRoot_Override() (gas: 53159)
[PASS] test_SetAccountRoot_WithOverrideControl() (gas: 57151)
Suite result: ok. 14 passed; 0 failed; 0 skipped; finished in 8.33ms (2.81ms CPU time)

Ran 12 tests for test/unit/withdrawals/VaultRootReceiver.t.sol:VaultRootReceiverTest
[PASS] test_EdgeCases() (gas: 85294)
[PASS] test_ErrorSelectors() (gas: 14253)
[PASS] test_InitialState() (gas: 24904)
[PASS] test_MultipleRoots_WithOverride() (gas: 80037)
[PASS] test_OverrideSetting_Change() (gas: 74863)
[PASS] test_RevertIf_SetVaultRoot_OverrideNotAllowed() (gas: 44862)
[PASS] test_RevertIf_SetVaultRoot_ZeroValue() (gas: 12882)
[PASS] test_SetVaultRoot_Initial() (gas: 43936)
[PASS] test_SetVaultRoot_Override() (gas: 53459)
[PASS] test_SetVaultRoot_WithOverrideControl() (gas: 57255)
[PASS] test_VaultRootSet_EventEmitted() (gas: 39997)
[PASS] test_VaultRootSet_EventEmitted_Override() (gas: 44720)
```

```
Suite result: ok. 12 passed; 0 failed; 0 skipped; finished in 8.46ms (2.93ms CPU time)

Ran 2 tests for test/proof-of-code/TestUninitializedStateVariables.sol:TestUninitializedStateVariables
[PASS] test_FunctionsExpectPreExistingState() (gas: 5339)
[PASS] test_VariablesArePrePopulatedFromExistingDeployment() (gas: 6176)
Suite result: ok. 2 passed; 0 failed; 0 skipped; finished in 347.47µs (45.82µs CPU time)

Ran 14 tests for test/unit/assets/BridgedTokenMapping.t.sol:BridgedTokenMappingTest
[PASS] test_GetTokenMapping_UnregisteredAsset() (gas: 27565)
[PASS] test_GetZKEVMAddressAndQuantum_UnregisteredAsset() (gas: 123162)
[PASS] test_IsMapped() (gas: 105380)
[PASS] test_RegisterToken_MultipleAssetsSameZKEVMAddress() (gas: 198872)
[PASS] test_RegisterToken_Native() (gas: 180372)
[PASS] test_RegisterToken_QuantumAtUpperBound() (gas: 103238)
[PASS] test_RevertIf_RegisterToken_AlreadyRegistered() (gas: 96082)
[PASS] test_RevertIf_RegisterToken_QuantumAboveBounds() (gas: 19614)
[PASS] test_RevertIf_RegisterToken_ZeroAddress() (gas: 18780)
[PASS] test_RevertIf_RegisterToken_ZeroAssetId() (gas: 17486)
[PASS] test_RevertIf_RegisterToken_ZeroQuantum() (gas: 17924)
[PASS] test_RevertIf_RegisterTokens_EmptyArray() (gas: 12142)
[PASS] test_registerTokenMappings_Multiple() (gas: 200410)
[PASS] test_registerTokenMappings_Single() (gas: 119059)
Suite result: ok. 14 passed; 0 failed; 0 skipped; finished in 1.52ms (1.21ms CPU time)

Ran 19 tests for test/unit/bridge/messaging/VaultRootReceiverAdapter.t.sol:VaultRootReceiverTest
[PASS] test_Constructor() (gas: 45475)
[PASS] test_Execute_EventEmitted() (gas: 150224)
[PASS] test_Execute_ValidMessage() (gas: 150755)
[PASS] test_RevertIf_Execute_AdapterStatePartiallySet() (gas: 57074)
[PASS] test_RevertIf_Execute_InvalidMessageCommand() (gas: 117626)
[PASS] test_RevertIf_Execute_InvalidPayloadTooShort() (gas: 116772)
[PASS] test_RevertIf_Execute_NotApprovedByGateway() (gas: 60612)
[PASS] test_RevertIf_Execute_UnauthorizedSenderAddress() (gas: 118446)
[PASS] test_RevertIf_Execute_UnauthorizedSenderChain() (gas: 115912)
[PASS] test_RevertIf_Execute_VaultRootReceiverNotSet() (gas: 86356)
[PASS] test_RevertIf_Execute_VaultRootSourceNotSet() (gas: 61305)
[PASS] test_RevertIf_SetVaultRootReceiver_Unauthorized() (gas: 17779)
[PASS] test_RevertIf_SetVaultRootReceiver_ZeroAddress() (gas: 15039)
[PASS] test_RevertIf_SetVaultRootSource_InvalidAddress() (gas: 16461)
[PASS] test_RevertIf_SetVaultRootSource_InvalidChain() (gas: 16223)
[PASS] test_RevertIf_SetVaultRootSource_Unauthorized() (gas: 20176)
[PASS] test_SetVaultRootReceiver() (gas: 47142)
[PASS] test_SetVaultRootSource() (gas: 85417)
[PASS] test_SetVaultRootSource_UpdateExisting() (gas: 96750)
Suite result: ok. 19 passed; 0 failed; 0 skipped; finished in 2.78ms (3.83ms CPU time)

Ran 20 tests for test/unit/bridge/starkex/StarkExchangeMigration.t.sol:StarkExchangeMigrationTest
[PASS] testToken() (gas: 3214)
[PASS] test_Initialize_ValidParameters() (gas: 2555386)
[PASS] test_MigrateHoldings_ERC20() (gas: 422800)
[PASS] test_MigrateHoldings_ETH() (gas: 87135)
[PASS] test_MigrateHoldings_Mixed() (gas: 465644)
[PASS] test_MigrateVaultRoot() (gas: 123086)
[PASS] test_RevertIf_Initialize_InvalidMigrationManager() (gas: 2444746)
[PASS] test_RevertIf_Initialize_InvalidRootSenderAdapter() (gas: 2442945)
[PASS] test_RevertIf_Initialize_InvalidWithdrawalProcessor() (gas: 2445747)
[PASS] test_RevertIf_Initialize_InvalidZkEVMBridge() (gas: 2442626)
[PASS] test_RevertIf_Initialize_Twice() (gas: 2541018)
[PASS] test_RevertIf_MigrateHoldings_EmptyArray() (gas: 21370)
[PASS] test_RevertIf_MigrateHoldings_ExcessBridgeFee() (gas: 413564)
[PASS] test_RevertIf_MigrateHoldings_InsufficientBridgeFee() (gas: 34502)
[PASS] test_RevertIf_MigrateHoldings_InsufficientETHBalance() (gas: 34077)
[PASS] test_RevertIf_MigrateHoldings_InsufficientTokenBalance() (gas: 42627)
[PASS] test_RevertIf_MigrateHoldings_Unauthorized() (gas: 33107)
[PASS] test_RevertIf_MigrateHoldings_ZeroAmount() (gas: 35214)
[PASS] test_RevertIf_MigrateHoldings_ZeroTokenAddress() (gas: 32922)
[PASS] test_RevertIf_MigrateVaultRoot_Unauthorized() (gas: 28163)
Suite result: ok. 20 passed; 0 failed; 0 skipped; finished in 4.32ms (7.30ms CPU time)

Ran 42 tests for test/unit/withdrawals/VaultWithdrawalProcessor.t.sol:VaultWithdrawalProcessorTest
[PASS] test_Constants() (gas: 18977)
[PASS] test_Constructor() (gas: 105988)
[PASS] test_Constructor_RootOverrideAllowed() (gas: 4061734)
[PASS] test_NonCriticalFunctions_WorkWhenPaused() (gas: 46469)
[PASS] test_PauseAndUnpause() (gas: 35868)
```

```
[PASS] test_ProcessValidEscapeClaim_ERC20() (gas: 758021)
[PASS] test_ProcessValidEscapeClaim_IMX() (gas: 433967)
[PASS] test_Receive() (gas: 20878)
[PASS] test_RegisterTokenMappings() (gas: 96575)
[PASS] test_RevertIf_AccountRootNotSet() (gas: 4565667)
[PASS] test_RevertIf_ClaimAlreadyProcessed() (gas: 492954)
[PASS] test_RevertIf_Constructor_InvalidAccountRootProviderOperator() (gas: 76718)
[PASS] test_RevertIf_Constructor_InvalidDefaultAdminOperator() (gas: 80030)
[PASS] test_RevertIf_Constructor_InvalidDisburserOperator() (gas: 78985)
[PASS] test_RevertIf_Constructor_InvalidPauserOperator() (gas: 78736)
[PASS] test_RevertIf_Constructor_InvalidTokenMappingManagerOperator() (gas: 77496)
[PASS] test_RevertIf_Constructor_InvalidUnpauserOperator() (gas: 79807)
[PASS] test_RevertIf_Constructor_InvalidVaultRootProviderOperator() (gas: 77393)
[PASS] test_RevertIf_Constructor_ZeroVaultVerifier() (gas: 87099)
[PASS] test_RevertIf_EmptyAccountProof() (gas: 230606)
[PASS] test_RevertIf_EmptyVaultProof() (gas: 100699)
[PASS] test_RevertIf_InsufficientBalance() (gas: 428821)
[PASS] test_RevertIf_InvalidAccountProof() (gas: 349762)
[PASS] test_RevertIf_InvalidVaultProof() (gas: 377854)
[PASS] test_RevertIf_Paused_Withdrawal() (gas: 254700)
[PASS] test_RevertIf_Receive_UnauthorisedSender() (gas: 21428)
[PASS] test_RevertIf_RegisterTokenMappings_Unauthorized() (gas: 21114)
[PASS] test_RevertIf_SetAccountRoot_Unauthorized() (gas: 18350)
[PASS] test_RevertIf_SetRootOverrideAllowed_NoChange() (gas: 28969)
[PASS] test_RevertIf_SetRootOverrideAllowed_Unauthorized() (gas: 19201)
[PASS] test_RevertIf_SetVaultRoot_AlreadySet_OverrideDisabled() (gas: 24757)
[PASS] test_RevertIf_SetVaultRoot_Unauthorized() (gas: 17295)
[PASS] test_RevertIf_UnauthorizedPause() (gas: 18537)
[PASS] test_RevertIf_UnauthorizedUnpause() (gas: 47585)
[PASS] test_RevertIf_UnauthorizedWithdrawal() (gas: 220403)
[PASS] test_RevertIf_UnregisteredAsset() (gas: 337506)
[PASS] test_RevertIf_VaultProofWrongLength() (gas: 143203)
[PASS] test_RevertIf_VaultRootNotSet() (gas: 4564643)
[PASS] test_RevertIf_ZeroAddress() (gas: 227166)
[PASS] test_SetAccountRoot() (gas: 27074)
[PASS] test_SetRootOverrideAllowed() (gas: 37484)
[PASS] test_SetVaultRoot() (gas: 18101)
Suite result: ok. 42 passed; 0 failed; 0 skipped; finished in 2.73s (1.09s CPU time)

Ran 5 tests for test/integration/bridge/starkex/StarkExchangeMigration.t.sol:StarkExchangeMigrationTest
[PASS] test_UpgradeStarkExchange() (gas: 2640555)
[PASS] test_UpgradeStarkExchange_VaultRootPreserved() (gas: 2643497)
[PASS] test_migrateVaultState() (gas: 2699090)
[PASS] test_migrate_ERC20Holdings() (gas: 3230523)
[PASS] test_migrate_ETHHoldings() (gas: 2810051)
Suite result: ok. 5 passed; 0 failed; 0 skipped; finished in 17.09s (17.25s CPU time)

Ran 2 tests for
 ↪  test/integration/proofs/withdrawals/VaultWithdrawalProcessor.t.sol:VaultWithdrawalProcessorIntegrationTest
[PASS] test_ProcessVaultWithdrawal_IMX() (gas: 2275254)
[PASS] test_ProcessVaultWithdrawal_USDC() (gas: 2594875)
Suite result: ok. 2 passed; 0 failed; 0 skipped; finished in 17.08s (29.46s CPU time)

Ran 20 tests for test/unit/verifiers/vaults/VaultEscapeProofVerifier.t.sol:VaultEscapeProofVerifierTest
[PASS] test_Constants() (gas: 11816)
[PASS] test_Constructor() (gas: 561620)
[PASS] test_ExtractDataFromMultipleProofs() (gas: 902726)
[PASS] test_ExtractLeafAndRootFromProof() (gas: 214454)
[PASS] test_ExtractLeafFromProof() (gas: 209021)
[PASS] test_ExtractRootFromProof() (gas: 184783)
[PASS] test_RevertIf_ExtractFunctions_WithLongProof() (gas: 150530)
[PASS] test_RevertIf_ExtractFunctions_WithOddLength() (gas: 67918)
[PASS] test_RevertIf_ExtractLeafAndRootFromInvalidProof() (gas: 28371)
[PASS] test_RevertIf_ExtractLeafFromInvalidProof() (gas: 27841)
[PASS] test_RevertIf_ExtractRootFromInvalidProof() (gas: 27168)
[PASS] test_RevertIf_VerifyProof_ExactlyAtLongLimit() (gas: 56592)
[PASS] test_RevertIf_VerifyProof_WithInvalidKey() (gas: 193971)
[PASS] test_RevertIf_VerifyProof_WithInvalidLength_Long() (gas: 56746)
[PASS] test_RevertIf_VerifyProof_WithInvalidLength_Odd() (gas: 27737)
[PASS] test_RevertIf_VerifyProof_WithInvalidLength_Short() (gas: 27183)
[PASS] test_RevertIf_VerifyProof_WithInvalidPath() (gas: 842761)
[PASS] test_VerifyMultipleValidEscapeProofs() (gas: 5347190)
[PASS] test_VerifyProof_MinimumValidLength() (gas: 2030283)
[PASS] test_VerifyValidEscapeProof() (gas: 1975281)
Suite result: ok. 20 passed; 0 failed; 0 skipped; finished in 17.10s (57.76s CPU time)
```

```
Ran 13 test suites in 17.12s (54.06s CPU time): 177 tests passed, 0 failed, 0 skipped (177 total tests)
```

8.2 Automated Tools

8.2.1 AuditAgent

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

9 About Nethermind

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

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

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

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

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

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

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

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

- **Voyager** is a user-friendly Starknet block explorer that offers comprehensive insights into the Starknet network. With its intuitive interface and powerful features, Voyager allows users to easily search for and examine transactions, addresses, and contract details. As an essential tool for navigating the Starknet ecosystem, Voyager is the go-to solution for users seeking in-depth information and analysis;

- **Horus** is an open-source formal verification tool for StarkNet smart contracts. It simplifies the process of formally verifying Starknet smart contracts, allowing developers to express various assertions about the behavior of their code using a simple assertion language;

- **Juno** is a full-node client implementation for Starknet, drawing on the expertise gained from developing the Nethermind Client. Written in Golang and open-sourced from the outset, Juno verifies the validity of the data received from Starknet by comparing it to proofs retrieved from Ethereum, thus maintaining the integrity and security of the entire ecosystem.

General Advisory to Clients

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

Disclaimer