Security Review Report
NM-0120 ZKX Starkway

NETHERMIND
SECURITY

(Dec 05, 2023)

Contents

1 Executive Summary

This document presents the security review performed by Nethermind on the ZKX Starkway. The reviewed contracts implement the Starkway, a mechanism that facilitates the transfer of ERC20 tokens between Ethereum and Starknet while also introducing distinctive features that enhance user interaction. The Starkway allows users to seamlessly transfer any ERC20 token from Ethereum to Starknet without requiring permission. If the token is not already present on Starknet, it is automatically deployed and initialized. Furthermore, the Starkway extends its functionality by permitting the execution of arbitrary calls by transmitting messages in conjunction with the deposit to Starknet. This opens up opportunities for the integration of Starkway in sophisticated cross-chain applications.

All fixes by the ZKX team were provided in the audit-fixes branch at commit 5564a902f10056d73d074789f00f5f5f9d69cdd4 which have been reviewed by Nethermind and merged into the main development branch.

The audit was performed using (a) manual analysis of the codebase, (b) automated analysis tools, (c) simulation of the smart contract, and (d) creation of test cases. **Along this document, we report** 26 points of attention, where nine are classified as `Low`, and sixteen are classified as `Best Practices` or `Informational`. 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 presents the system overview. Section 5 discusses the risk rating methodology adopted for this audit. Section 6 details the issues. Section 7 discusses the documentation provided by the client for this audit. Section 8 presents the compilation, tests, and automated tests. Section 9 concludes the document.

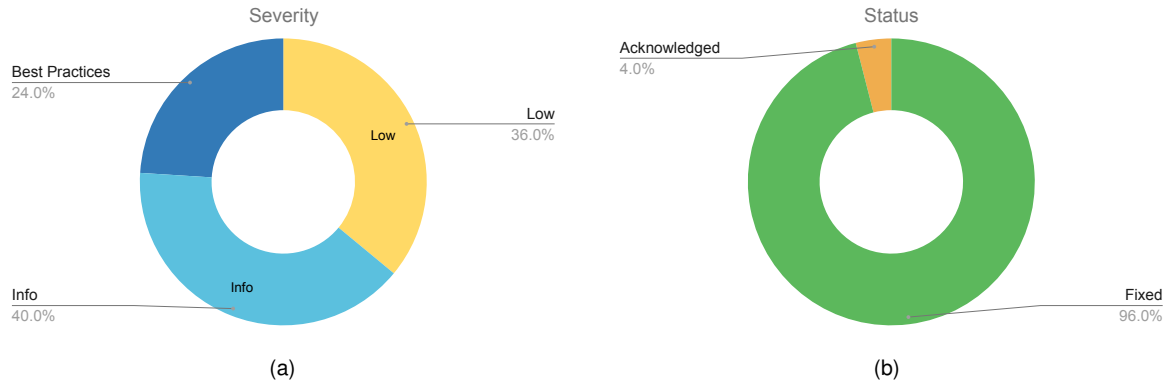

Fig. 1: Distribution of issues: Critical (0), **High** (0), **Medium** (0), **Low** (9), **Undetermined** (0), **Informational** (10), **Best Practices** (6). **Distribution of status: Fixed** (24), **Acknowledged** (1), **Mitigated** (0), **Unresolved** (0), **Partially Fixed** (0)

Summary of the Audit

Audit Type	Security Review
Initial Report	Sep. 27, 2023
Response from Client	Nov. 17, 2023
Final Report	Dec. 05, 2023
Methods	Manual Review, Automated Analysis
Repository	starkway_v2
Commit (Audit)	f933f5980bf3fd197cdf40b17b15711fe79fae90
Branch (Fix Review)	audit-fixes
Commit (Final)	5564a902f10056d73d074789f00f5f5f9d69cdd4
Documentation Assessment	Medium
Test Suite Assessment	Medium

2 Contracts

2.1 Solidity contracts

	Contract	LoC	Comments	Ratio	Blank	Total
1	L1/contracts/interfaces/IStarkwayAggregate.sol	6	3	50.0%	2	11
2	L1/contracts/interfaces/IStarkwayHelper.sol	14	19	135.7%	3	36
3	L1/contracts/interfaces/starkway/IStarkway.sol	75	76	101.3%	12	163
4	L1/contracts/interfaces/starkway/IStarkwayEvents.sol	48	43	89.6%	8	99
5	L1/contracts/interfaces/starkway/IStarkwayErrors.sol	23	41	178.3%	21	85
6	L1/contracts/interfaces/starkway/IStarkwayAuthorized.sol	44	51	115.9%	9	104
7	L1/contracts/interfaces/multiconnectable/IMultiConnectableErrors.sol	7	5	71.4%	5	17
8	L1/contracts/interfaces/multiconnectable/IMultiConnectableStateInfo.sol	13	4	30.8%	4	21
9	L1/contracts/interfaces/multiconnectable/IMultiConnectableEvents.sol	7	5	71.4%	5	17
10	L1/contracts/interfaces/multiconnectable/IMultiConnectable.sol	7	1	14.3%	2	10
11	L1/contracts/interfaces/vault/IStarkwayVaultAuthorized.sol	16	44	275.0%	12	72
12	L1/contracts/interfaces/vault/IStarkwayVault.sol	43	57	132.6%	18	118
13	L1/contracts/interfaces/starknet/IStarknetMessaging.sol	62	1	1.6%	13	76
14	L1/contracts/interfaces/starknet/IStarknetMessagingEvents.sol	50	7	14.0%	9	66
15	L1/contracts/implementation/StarkwayHelper.sol	124	14	11.3%	18	156
16	L1/contracts/implementation/StarkwayVault.sol	216	73	33.8%	45	334
17	L1/contracts/implementation/Starkway.sol	702	87	12.4%	76	865
18	L1/contracts/implementation/helpers/TokenUtils.sol	74	1	1.4%	10	85
19	L1/contracts/implementation/helpers/Constants.sol	11	8	72.7%	4	23
20	L1/contracts/implementation/helpers/FeltUtils.sol	26	1	3.8%	6	33
21	L1/contracts/implementation/base_contracts/MultiConnectable.sol	101	45	44.6%	32	178
22	L1/contracts/implementation/base_contracts/PairedToL2.sol	40	19	47.5%	17	76
	Total	**1709**	**605**	**35.4%**	**331**	**2645**

2.2 Cairo contracts

	Contract	LoC	Comments	Ratio	Blank	Total
1	L2/src/utils.cairo	1	0	0.0%	0	1
2	L2/src/starkway.cairo	1009	297	29.4%	177	1483
3	L2/src/datatypes.cairo	94	4	4.3%	15	113
4	L2/src/bridge_adapters.cairo	1	0	0.0%	0	1
5	L2/src/interfaces.cairo	186	30	16.1%	13	229
6	L2/src/lib.cairo	11	0	0.0%	0	11
7	L2/src/libraries.cairo	2	0	0.0%	0	2
8	L2/src/starkway_helper.cairo	151	57	37.7%	39	247
9	L2/src/admin_auth.cairo	128	43	33.6%	32	203
10	L2/src/plugins.cairo	5	0	0.0%	1	6
11	L2/src/erc20.cairo	1	0	0.0%	0	1
12	L2/src/erc20/erc20.cairo	222	20	9.0%	39	281
13	L2/src/utils/helpers.cairo	69	15	21.7%	15	99
14	L2/src/bridge_adapters/starkgate_adapter.cairo	32	0	0.0%	4	36
15	L2/src/libraries/fee_library.cairo	100	12	12.0%	19	131
16	L2/src/libraries/reentrancy_guard.cairo	19	6	31.6%	4	29
	Total	**2031**	**484**	**23.8%**	**358**	**2873**

3 Summary of Issues

	Finding	Severity	Update
1	Any token initialization can be interrupted requiring admin action	Low	Acknowledged
2	Custom fee segments do not cover all deposit range	Low	Fixed
3	Deposit with initialization of tokens depends on transaction ordering	Low	Fixed
4	Inconsistent validation between L1 and L2 contracts during fee setting	Low	Fixed
5	Interpretation of maximum range is inconsistent	Low	Fixed
6	Privileged address change should use "set-then-claim" approach	Low	Fixed
7	The `withdraw_admin_fees` method updates `fee_withdrawn` storage variable without checking total collected fees	Low	Fixed
8	The storage array `allConnectedTargets` is never written to	Low	Fixed
9	Tokens without a `withdrawal_range` cannot be bridged back to Ethereum by default	Low	Fixed
10	Admin can initiate self-removal	Info	Fixed
11	Decimals are not considered during in `withdraw_multi(...)`	Info	Fixed
12	Deposit message contents are not validated to fit within a `felt`	Info	Fixed
13	Implement camel-case methods for ERC20 functions	Info	Fixed
14	Inconsistent spelling for "initialize" across contracts	Info	Fixed
15	Inconsistent zero amount checks between L2 deposit functions	Info	Fixed
16	Library `FeltUtils` declares duplicate constant	Info	Fixed
17	The `admin_auth` constructor allows same address for both admins	Info	Fixed
18	The protocol admin management mechanism security varies between the layers	Info	Fixed
19	Zero address check exists only for L1 ETH transfers	Info	Fixed
20	Check bridge interface before writing to storage.	Best Practices	Fixed
21	Unnecessary storage mapping `everConnectedAsStarknet`	Best Practices	Fixed
22	Unnecessary traits are derived for custom types	Best Practices	Fixed
23	Unnecessary usage of storage slot	Best Practices	Fixed
24	Use `BoundedInt` to represent unsigned integer max values	Best Practices	Fixed
25	Using different versions of Cairo syntax	Best Practices	Fixed

4 System Overview

The Starkway protocol can be split into the Ethereum and Starknet layers. It enables users to transfer ERC20 tokens from Ethereum to Starknet permissionless. If the token has not been deployed on Starknet, the protocol automatically initiates the deployment and initialization of the token. Alongside the deployment of funds, Starkway also allows for transmitting arbitrary call messages to Layer 2 (L2), thereby facilitating complex cross-chain interactions. The protocol is not limited to bridging tokens back to Ethereum for native (Starkway) tokens alone, but it also extends this functionality to assets from other Starknet bridges through adapters. Below, we show the contract interaction graphs for both layers and describe each contract and its purpose.

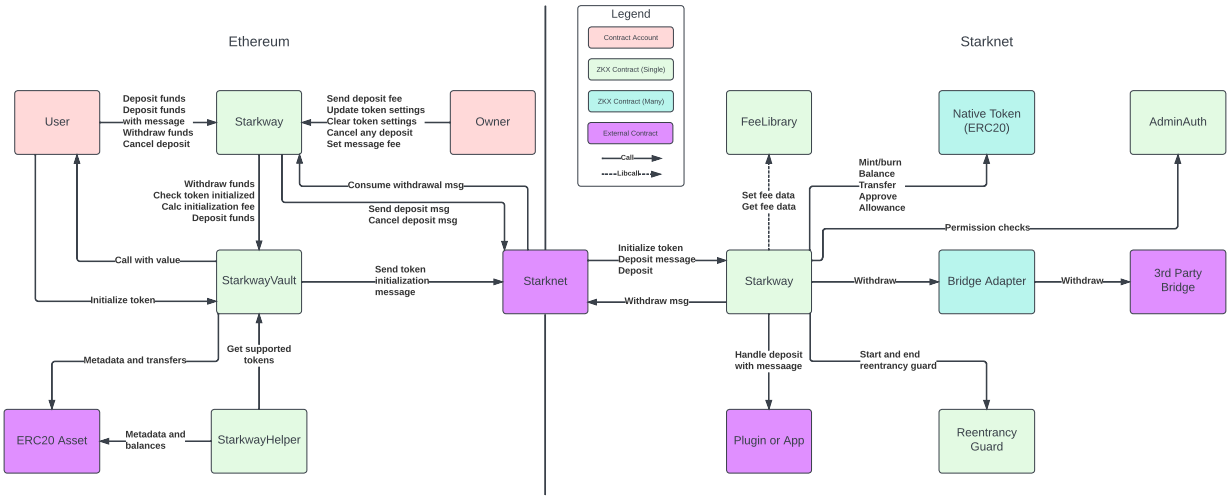

Fig. 2: Starkway - Structural Diagram of contracts on both layers

4.1 Ethereum layer overview

Starkway Main L1 entrypoint which users interact with. Allows users to deposit, send, and withdraw funds by consuming L2 messages.

StarkwayVault Holds all the bridged funds and initializes new tokens on Starknet. This is the only part of the protocol that will not be upgraded.

StarkwayHelper Provides tokens metadata for `StarkwayVault`.

User Any address that interacts with the protocol.

Owner Special address that sets parameters and can deposit L2 message in critical cases.

ERC20 Asset An external `ERC20` asset used for round asset deposits.

Starknet The `Starknet` contract on `Ethereum` which is used to communicate with the `Starknet` chain.

4.2 Starknet layer overview

Starkway Main L2 entrypoint which users interact with. Allows funds withdrawal with native (Starkway) tokens or other supported tokens. Sends a message to L1, which then must be consumed.

FeeLibrary Provides functionality to set and read fees.

ReentrancyGuard Provides functionality to prohibit reentrancy.

AdminAuth Manages the admins of the protocol. Two different admins must confirm each addition and removal.

Native Token (ERC20) Tokens deployed by the `Starkway` bridge.

Bridge Adapter Those contracts allow for communication sending non-native tokens to the specified bridge.

Third Party Bridge Receives tokens from Starkway through the adapter and bridges the assets to Ethereum.

Plugin or App Executes the message sent alongside the deposit.

5 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 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 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 [Low] Any token initialization can be interrupted requiring admin action

File(s): StarkwayVault.sol

Description: The Starkway protocol allows non-privileged users to initialize a token in two ways: Through a deposit or by manually calling `initToken(...)`.

The function `initToken(...)` is the most direct initialization method, where you pass an address, and an initialize message is sent to the associated Starkway L2 contract. One part of the initialization checks is to ensure that a token has not already been initialized by checking the `initStatusByToken` mapping; however, this does not guarantee that the Starkway L2 contract processed the initialization message.

A malicious user may call `initToken(...)` with a `msg.value` of 1 Wei, which will be too low for any sequencer to accept. The `StarkwayVault` contract will set the `initStatusByToken` value to `TOKEN_IS_INITIALIZED` for the given token, even though the sequencers will never process the message on Starknet. The relevant functions are shown below:

```
function initToken(address token) external payable {
  if (initStatusByToken[token] == TOKEN_IS_INITIALIZED) {
    revert StarkwayVault__TokenAlreadyInitialized();
  }

  (string memory name, string memory symbol, uint8 decimals) = _getMetadata(token);

  ////////////////////////////////////////////////////////////////////////////////
  // @audit-issue Attacker can pass 1 Wei msg.value and sequencer will never process on L2
  //              The token will be considered initialized on L1, but not on L2
  //              It will not be possible to re-initialize due to the init check above
  ////////////////////////////////////////////////////////////////////////////////

  _performInitialization({
    token: token,
    name: name,
    symbol: symbol,
    decimals: decimals,
    initFee: msg.value
  });
}
```

This causes an inconsistency between layers, where L1 considers the token initialized, but L2 does not. As such, any deposit will succeed on L1 but revert when processed on L2. This can be done on any token that has not yet been initialized.

Once the attack is done, any further attempts to initialize the token will revert due to the already-initialized validation mentioned above, requiring the token to be manually initialized on Starkway L2 by an admin.

Recommendation(s): Consider adding validation in `_performInitialization(...)` to ensure that the `initFee` is greater than or equal to the storage variable `initMessageFee`.

Status: Acknowledged

Update from the client: We acknowledge this issue, but there's no simple fix to it. Starknet messaging fee can't be calculated on-chain and there's no reasonable minimum fee value to be hardcoded in the contract to solve the issue. There's also a admin-controlled mechanism on L2 to perform token initialization if the init message fails to be consumed on the L2 side. We are going to execute a script to initialize 100+ most widely used tokens which will prevent this problem from happening.

6.2 [Low] Custom fee segments do not cover all deposit range

File(s): Starkway.sol

Description: The function _validateTokenSettings(...) checks if provided fee segments are defined correctly. One of the responsibilities of this function is ensuring the space of all allowed deposit amounts is covered by the fee segments, which is done with the following requirement statement:

```
if (maxDeposit > prevToAmount && prevToAmount != 0) revert InvalidMaxDeposit();
```

The above statement checks that the last segment is within a space of allowed deposit amounts. However, the value of maxDeposit can be set as zero, which is interpreted as an unlimited amount of deposit. This allows for avoiding the revert if maxDeposit = 0 and prevToAmount != 0 because the statement maxDeposit > prevToAmount returns false, even though the maxDeposit == 0 means unlimited deposit. Consequently, it would allow for a deposit that would not be covered by any fee segment and result in reverting in the function _resolveDepositFeeRate(...).

Recommendation(s): Consider ensuring that the segments cover all possible deposit amounts by taking maxDeposit = 0 into account in the function _validateTokenSettings(...).

Status: Fixed

Update from the client: Addressed the issue in the commit 04d46b0. Also added a test for this case: b4856aa.

6.3 [Low] Deposit with initialization of tokens depends on transaction ordering

File(s): Starkway.sol, StarkwayVault.sol, starkway.cairo

Description: The Starkway bridge allows users to deposit funds from Ethereum to Starknet and initialize a bridged token if it hasn't been deployed yet on L2. While on Ethereum, those two actions are done in one transaction, the Starknet part of a bridge receives two messages that trigger the two methods: initialize_token(...) and deposit(...). It is crucial that the initialization is executed before the deposit. However, the order of execution of submitted messages is not ensured by the order of submission. Currently, Starknet operates on just one centralized Sequencer that executes messages by order of submission. This will most likely change since the block creation and execution process is planned to be decentralized. Then, the order of execution would not be ensured by the order of submission. If the deposit message is executed before the initialization, the funds won't be deposited, and the user will have to cancel the message on L1 manually.

Recommendation(s): Consider avoiding the dependence on transaction order for cross-layer messages. One possible way to mitigate this issue is sending one message to the L2 starkway contract that would both initialize and deposit the token.

Status: Fixed

Update from the client: We decided to decouple init & deposit messages. It was the most challenging issue from the design perspective. We've considered a solution when for a deposit of an uninitialized token init and deposit L1-to-L2 messages would be merged together and sent to Starknet as a single message. But this solution required too many design changes since it's StarkwayVault who sends init message and is responsible for token initialization, but deposit message is sent by Starkway. In that case the only available option was appending deposit payload to init message. It didn't seem to be a good solution: it required changes on L2 side, complicated the architecture, went against segregation of responsibilities and could introduce problems for integration of the Vault with the next versions of Starkway contract.

With that in mind, we decided to decouple init and deposit operations. It means that for a successful deposit a token must be initialized already, otherwise the deposit's going to fail. We don't expect this limitation to significantly affect the user experience. Firstly, a token has to be initialized only once, and secondly, we're going to execute a script initializing 100+ most widely used tokens. This solution also required minimum changes on L1 and no changes on L2 side.

Addressed the issue in 3042d80.

6.4 [Low] Inconsistent validation between L1 and L2 contracts during fee setting

File(s): fee_library.cairo, Starkway.sol

Description: Custom fee segments are enabled on both L1 and L2 contracts. However, there are significant inconsistencies in the validation of provided fee segments, which we list below:

- the fee rate should be lower for higher deposit/withdraw amounts. This property is checked during segments setting only in the L2 fee_library. The L1 contract does not check for this property;
- both layers have a maximum fee rate set to 300. However, only L1 Starkway contract validates that the fee rates in segments are less or equal to MAX_FEE_RATE. L2 contracts allow higher fee rates in the segments;

Additionally, L1 interprets the maximum fee range 0 as infinity, whereas L2 interprets 0 as zero.

Recommendation(s): Consider unifying the verification of fee setting.

Status: Fixed

Update from the client: L2 inconsistency addressed in 2cfd174. We now check segment fee rate <= MAX_FEE_RATE and also treat 0 as infinity on L2. L1 inconsistency addressed in 04d46b0 and a test for it added in 69b1fb8.

6.5 [Low] Interpretation of maximum range is inconsistent

File(s): Starkway.sol, starkway.cairo

Description: The maximum range of withdrawal on the L2 `starkway` can be set as `0`, which is interpreted as unlimited during the set:

```
fn set_withdrawal_range(
    ref self: ContractState, l1_token_address: EthAddress, withdrawal_range: WithdrawalRange
) {
    self._verify_caller_is_admin();
    let native_token_address: ContractAddress = self
        .native_token_l2_address
        .read(l1_token_address);
    assert(native_token_address.is_non_zero(), 'SW: Token uninitialized');
    let zero: u256 = u256 { low: 0, high: 0 };
    // @audit-note 0 value maximum withdrawal range is valid even if the value of the minimum range is non-zero
    if (withdrawal_range.max != zero) {
        assert(withdrawal_range.min < withdrawal_range.max, 'SW: Invalid min and max');
    }
    self.withdrawal_ranges.write(l1_token_address, withdrawal_range);
}
```

However, when the withdrawal amount is validated, the value `0` of the maximum range is interpreted not as an infinite range but as `0`:

```
fn _verify_withdrawal_amount(
    self: @ContractState, l1_token_address: EthAddress, withdrawal_amount: u256
) {
    let withdrawal_range = self.withdrawal_ranges.read(l1_token_address);
    let safety_threshold = withdrawal_range.max;
    // @audit-note the 0 value of maximum withdrawal is not interpreted as infinity
    assert(withdrawal_amount < safety_threshold, 'SW: amount > threshold');
    let min_withdrawal_amount = withdrawal_range.min;
    assert(min_withdrawal_amount <= withdrawal_amount, 'SW: min_withdraw > amount');
}
```

This creates an inconsistency in setting and validating the maximum range within the L2 `starkway` contract. Moreover, the validation of the maximum range on L2 is inconsistent with the validation on L1 since L1 `Starkway` interprets the maximum range `0` as infinity.

Recommendation(s): Consider unifying the interpretation of the maximum range between L1 and L2 and within `starkway` contract between setting and validating the maximum range.

Status: Fixed

Update from the client: Addressed in b6d429c. We now interpret 0 as infinity on Starkway L2.

6.6 [Low] Privileged address change should use "set-then-claim" approach

File(s): starkway.cairo

Description: The function `set_admin_auth_address(...)` is used to change the `admin_auth_address`, which has full control over the L2 Starkway contract. If this address is set incorrectly, it may lead to the protocol admin features being left in an unreachable state; it will not be possible to change the address again due to the caller admin requirement, as shown below:

```
fn set_admin_auth_address(ref self: ContractState, admin_auth_address: ContractAddress) {
    ////////////////////////////////////////////////////////////////////////////////
    // @audit-issue One time address change with no validations
    //              A set-then-claim would prevent irrersible accidental changes
    ////////////////////////////////////////////////////////////////////////////////
    self._verify_caller_is_admin();
    self.admin_auth_address.write(admin_auth_address);
}
```

Recommendation(s): Consider using a "set-then-claim" approach, where a new admin address is proposed, and then they must call some function to claim ownership.

Status: Fixed

Update from the client: Addressed in 0a9566f. Admin can propose a new address for `admin_auth` and the new address has to claim the ownership by calling a separate function.

6.7 [Low] The `withdraw_admin_fees` method updates `fee_withdrawn` storage variable without checking total collected fees

File(s): `starkway.cairo`

Description:

The function `withdraw_admin_fees(...)` is an admin-only function that is used to withdraw feels collected by the Starkway L2 contract. As part of the fee withdrawal logic, the amount of remaining fees available to withdraw is calculated, as shown below:

```
fn withdraw_admin_fees(...) {
    //...

    let current_total_fee_collected = self.total_fee_collected.read(l1_token_address);
    let current_fee_withdrawn = self.fee_withdrawn.read(l1_token_address);
    ////////////////////////////////////////////////////////////////////////////////////
    // @audit-issue `current_fee_withdrawn` can be greater than collected amount if direct transfer
    ////////////////////////////////////////////////////////////////////////////////////
    let net_fee_remaining = current_total_fee_collected - current_fee_withdrawn;

    //...
}
```

It is possible for a user to manually send tokens to the contract outside of deposit logic, which will mean that while the tokens belong to the Starkway L2 contract, no fee storage data is updated. If an admin withdrew their available fees plus the directly transferred tokens, the `current_fee_withdrawn` will be greater than the `current_total_fee_collected`, since the extra direct transfer is withdrawn but was never considered to be fees that were collected.

In this situation, for following withdrawals the admin must wait until `current_total_fee_collected` has accumulated enough to be greater than `current_fee_withdrawn`.

It should be noted that this issue is known since a commend related to this behavior exists, but the logic to address the issue has been commented:

```
// Below case would be triggered, if transfer of tokens happen from outside
// (which is not through deposit). Below condition prevents admin from withdrawing those tokens.
// Commenting the condition for withdrawal to happen
// assert(withdrawal_amount <= net_fee_remaining, 'SW:Amount exceeds fee remaining');
```

Recommendation(s): Consider checking `fee_withdrawn` is less than `total_fee_collected` before updating storage.

Status: Fixed

Update from the client: Addressed in 0879f93. The admin can withdraw any amount of fees (and extra tokens) if it has the relevant token balance irrespective of amount of fees collected. However, as pointed out in the issue, we now update `fee_withdrawn` after ensuring it cannot be greater than `total_fee_collected`. This ensures that `fee_withdrawn` is always <= `total_fee_collected`.

6.8 [Low] The storage array `allConnectedTargets` is never written to

File(s): `MultiConnectable.sol`

Description: The abstract contract `MultiConnectable` defines an array `allConnectedTargets` which should store an array of connected `Starkway` contracts. It is used by two functions:

- `getAllConnections(...)`: Lists all targets that have ever connected, including currently disconnected targets;
- `_disconnect(...)`: Disconnects a currently connected target, ensuring at least one connected version remains;

However the array `allConnectedTargets` is never written to, and will always remain empty. For the function `getAllConnections(...)` this will cause an empty array to be returned on every call, as shown below:

```
function getAllConnections() external view returns (ConnectionInfo[] memory) {
  uint256 length = allConnectedTargets.length;
  ConnectionInfo[] memory result = new ConnectionInfo[](length);

  ////////////////////////////////////////////////////////////////
  // @audit-issue Length will always be zero, result will be empty
  ////////////////////////////////////////////////////////////////
  for (uint i; i < length;) {
    address target = allConnectedTargets[i];
    uint256 status = statusByTarget[target];
    uint256 statusDate = statusDateByTarget[target];
    result[i] = ConnectionInfo({
      target: target,
      status: status,
      statusDate: statusDate
    });
    unchecked { ++i; }
  }

  return result;
}
```

For the function `_disconnect(...)`, the for loop will never execute since the length of the array is zero. If the loop does not run then `activeConnectionsCount` will not increment, causing the active connection check to fail. A snippet from this function is shown below:

```
function _disconnect(address target) internal {
  _checkConnectionStatus(target, STATUS_CONNECTED);
  uint256 totalConnectionsCount = allConnectedTargets.length;
  uint256 activeConnectionsCount;
  for (uint256 i; i < totalConnectionsCount;) {
    address targetAtIndex = allConnectedTargets[i];
    uint256 status = statusByTarget[targetAtIndex];
    if (status == STATUS_CONNECTED) {
      unchecked { ++activeConnectionsCount; }
    }
    unchecked { ++i; }
  }

  ////////////////////////////////////////////////////////////////////
  // @audit-issue For loop will never execute, so this will always revert
  ////////////////////////////////////////////////////////////////////
  if (activeConnectionsCount < 2) {
    revert MultiConnectable__MustRemainConnectedVersion();
  }

  ...
}
```

Recommendation(s): The storage array `allConnectedTargets` should be written to when a new target is connected.

Status: Fixed

Update from the client: It could become a nasty issue and could be discovered too late: when connecting a new version of Starkway of disconnecting the old one. Good catch. The issue is addressed in 6cb6571 and extensive tests covering connection-disconnection process are added in 2f57d06.

6.9 [Low] Tokens without a `withdrawal_range` cannot be bridged back to Ethereum by default

File(s): `starkway.cairo`

Description: When withdrawing tokens from Starknet to Ethereum, the internal function _verify_withdrawal_amount(...) is called to ensure that the withdrawal amount is within the specified min and max range. When the max range for a given token is unset (zero), no u256 value will satisfy the safety threshold assert statement, causing every withdrawal attempt from Starknet to Ethereum to fail. The relevant code is shown below:

```
fn _verify_withdrawal_amount(
    self: @ContractState, l1_token_address: EthAddress, withdrawal_amount: u256
) {
    let withdrawal_range = self.withdrawal_ranges.read(l1_token_address);
    let safety_threshold = withdrawal_range.max;
    ////////////////////////////////////////////////////////////////////////////
    // @audit-issue When the withdrawal range is not set, the max is zero
    //              When max threshold is zero, no u256 value will satisfy the assert
    ////////////////////////////////////////////////////////////////////////////
    assert(withdrawal_amount < safety_threshold, 'SW: amount > threshold');
    let min_withdrawal_amount = withdrawal_range.min;
    assert(min_withdrawal_amount <= withdrawal_amount, 'SW: min_withdraw > amount');
}
```

This effectively creates a permissioned withdrawal system, where the L2 Starkway admins have to manually specify a non-zero withdrawal range to allow funds to be bridged back to L1.

Recommendation(s): Consider changing the `_verify_withdrawal_amount(...)` function to behave similarly to the Ethereum contracts, where the zero value is treated as a no-limit withdrawal. If this permissioned behavior is intended, consider creating separate, specific logic related to locking and unlocking token withdrawals rather than integrating this logic into withdrawal ranges.

Status: Fixed

Update from the client: Addressed in commit e4f23a7. We now treat 0 for `withdrawal_range.max` as infinity. Hence, if withdrawal range has not been set, it will not stop withdrawal of any token. We also implement a permissioned withdrawal system based on the L2 token address (which is enabled by default for all tokens). For withdraw function, the L2 token should have withdrawal allowed. For `withdraw_multi` function all the tokens being withdrawn should have withdrawal allowed. Apart from this, `can_withdraw_multi` also checks that all tokens that a user wants to withdraw have withdrawal enabled. The `prepare_withdrawal_lists` function now considers only those tokens which have withdrawal enabled. The `init_token` and `whitelist_token` functions allow withdrawal for any token being initialized or whitelisted.

6.10 [Info] Admin can initiate self-removal

File(s): `admin_auth.cairo`

Description: The function `_update_admin_mapping(...)` adds or removes an admin from a `admin_lookup(...)` mapping. The process requires two distinct admins, other than the managed address, to submit an action of removal or adding an address. However, the admin currently can initiate removal action of its own address, since there is no check if the caller is not the same address as the managed address.

Recommendation(s): Consider checking if the caller is not providing its own address when managing admin access.

Status: Fixed

Update from the client: Addressed in b2fbdd4.

6.11 [Info] Decimals are not considered during in `withdraw_multi(...)`

File(s): `starkway.cairo`

Description: The multi-withdrawal allows for withdrawing through Starkway bridge different L2 tokens that point to the same L1 token. The function `withdraw_multi(...)` sums the amounts of provided tokens. If a token (native or non-native) with enough liquidity is found, it is used to bridge funds to L1. However, the `withdraw_multi(...)` function does not consider the decimals during the calculation of all used L2 tokens. We present the abovementioned code below:

```
fn _calculate_withdrawal_amount(
    self: @ContractState,
    transfer_list: @Array<TokenAmount>,
    l1_token_address: EthAddress,
    native_l2_address: ContractAddress,
) -> u256 {
    let transfer_list_len = transfer_list.len();
    let mut index = 0_u32;
    let mut amount = u256 { low: 0, high: 0 };
    loop {
        if (index == transfer_list_len) {
            break ();
        }

        if (*transfer_list.at(index).l2_address != native_l2_address) {
            let token_details: L2TokenDetails = self
                .whitelisted_token_details
                .read(*transfer_list.at(index).l2_address);
            // check that all tokens passed for withdrawal represent same l1_token_address
            assert(token_details.l1_address == l1_token_address, 'SW: L1 address Mismatch');
        }

        assert(
            *transfer_list.at(index).l1_address == l1_token_address,
            'SW: Incompatible L1 addr'
        );
        // @audit-note amount is added without taking decimals into account
        amount += *transfer_list.at(index).amount;
        index += 1;
    };
    amount
}
```

Consequently, with inconsistent decimals, the user or bridge may lose funds due to incorrect calculation.

Recommendation(s): Consider verifying that all the tokens have the same number of decimals. Alternatively, calculate the token amount in `_calculate_withdrawal_amount(...)` using the token's decimals. While this scenario is rather unlikely, we strongly recommend verifying instead of trusting in third-party correctness.

Status: Fixed

Update from the client: Addressed in bee0be9. Also fixes "Inconsistent zero-amount checks between L2 deposit functions".

6.12 [Info] Deposit message contents are not validated to fit within a `felt`

File(s): `Starkway.sol`

Description: When using the function `depositFundsWithMessage(...)`, the caller can specify a particular Starknet contract address that will receive a call with the data from the argument `messagePayload`. However, the contents of the `messagePayload` are not validated to be within the range of a `felt` type and may lead to messages not being consumed. A snippet from the payload builder function is shown below:

```
function _buildDepositWithMessagePayload(...) private pure returns (uint256[] memory) {
  uint256 msgLength = messagePayload.length;
  uint256[] memory payload = new uint[](9 + msgLength);
  (uint256 depositLow, uint256 depositHigh) = FeltUtils.splitIntoLowHigh(deposit);
  (uint256 feeLow, uint256 feeHigh) = FeltUtils.splitIntoLowHigh(depositFee);

  payload[0] = uint256(uint160(token));
  payload[1] = uint256(uint160(senderAddressL1));
  payload[2] = recipientAddressL2;
  payload[3] = depositLow;
  payload[4] = depositHigh;
  payload[5] = feeLow;
  payload[6] = feeHigh;

  // Append message
  payload[7] = messageRecipientL2;
  payload[8] = msgLength;
  for (uint256 i; i < msgLength; ) {

    ////////////////////////////////////////////////////////////////////////
    // @audit-issue Each entry in the message payload should fit within a felt
    ////////////////////////////////////////////////////////////////////////

    payload[9 + i] = messagePayload[i];
    unchecked { ++i; }
  }
  return payload;
}
```

Recommendation(s): Consider validating the message payload in `_buildDepositWithMessagePayload(...)`.

Status: Fixed

Update from the client: The issue was addressed in 956a187. Tests for the issue were added in 90119fb.

6.13 [Info] Implement camel-case methods for ERC20 functions

File(s): `erc20.cairo`

Description: The ERC20 implementation used by Starkway native tokens has function names with snake-case, following the convention of Rust and Rust-like languages. However, most of the ERC20 tokens existing in the Starknet ecosystem still use camel-case naming. The Starkway L2 contract implementation assumes that all token functions will be in snake-case, meaning that calls to any non-native token with a camel-case name will revert.

Recommendation(s): Consider developing a new ERC20 implementation that supports both camel-case and snake-case methods to provide better interoperability.

Status: Fixed

Update from the client: Addressed in a38d39f - we have added support for camelCase functions in the native ERC20 contract and included a field for camelCase (`is_erc20_camel_case`) in the L2TokenDetails struct. Interaction is done through camel case functions for non-native tokens which have this field set to true.

6.14 [Info] Inconsistent spelling for "initialize" across contracts

File(s): Starkway.cairo

Description: The function `_init_token(...)` emits an event with the name "INITIALISE", where every other reference to a token being initialized using the spelling "INITIALIZE" (functions, comments, revert messages). The event name and emission are shown below:

```
let mut keys = ArrayTrait::new();
keys.append(l1_token_address.into());
keys.append(token_details.name);
keys.append('INITIALISE');
let mut data = ArrayTrait::new();
data.append(contract_address.into());

emit_event_syscall(keys.span(), data.span());
```

Recommendation(s): Consider ensuring consistency between all contracts by changing "INITIALISE" to "INITIALIZE".

Status: Fixed

Update from the client: Addressed in 2f137ba.

6.15 [Info] Inconsistent zero amount checks between L2 deposit functions

File(s): starkway.cairo

Description: The `starkway` L2 contract features two deposit functions: `deposit` and `depositWithMessage`. These functions have inconsistent input validation, where `depositWithMessage(...)` ensures that the deposit amount cannot be zero, while `deposit(...)` does not.

Recommendation(s): Consider adding a zero amount validation to the `deposit(...)` function to improve consistency between both deposit methods.

Status: Fixed

Update from the client: Addressed in bee0be9. Also fixes “Decimals are not considered during in `withdraw_multi`”.

6.16 [Info] Library `FeltUtils` declares duplicate constant

File(s): FeltUtils.sol

Description: The library `FeltUtils` defined a constant `LOW_BITS_MASK`, which is used to help split a `uint256` value into the `high` and `low` bits to assist with Starknet messaging. This constant is already defined in `Constants.sol` and can be imported, rather than defined again.

Recommendation(s): Consider removing the unnecessary `LOW_BITS_MASK` declaration in `FeltUtils` and instead import the value from `Constants.sol`

Status: Fixed

Update from the client: Removed the unnecessary constant in 580a57e.

6.17 [Info] The `admin_auth` constructor allows same address for both admins

File(s): `admin_auth.cairo`

Description: When deploying the `admin_auth` contract, the caller must specify two addresses which will have admin privileges. The constructor does not ensure that these addresses are not the same. If the same address is passed for both arguments, it will not be possible to make any further changes to admin privileges using `add_admin(...)` or `remove_admin(...)` since another admin with a different address is required. The constructor is shown below:

```
#[constructor]
fn constructor(... , admin_1: ContractAddress, admin_2: ContractAddress) {
    ////////////////////////////////////////////////////////////////////
    // @audit-issue Can have the same address for `admin_1` and `admin_2`
    ////////////////////////////////////////////////////////////////////
    self.admin_lookup.write(admin_1, true);
    self.admin_lookup.write(admin_2, true);
    self.min_num_admins.write(2_u8);
    self.current_total_admins.write(2_u8);

    self.emit(Event::AdminAdded(AdminAdded { address: admin_1 }));
    self.emit(Event::AdminAdded(AdminAdded { address: admin_2 }));
}
```

Recommendation(s): Consider adding a check to the constructor of `admin_auth` to ensure that the two admin addresses are not the same.

Status: Fixed

Update from the client: Addressed through 4a4dcfc. We now check that admin addresses are not zero and not the same.

6.18 [Info] The protocol admin management mechanism security varies between the layers

File(s): `admin_auth.cairo`, `Starkway.sol`

Description: The L1 `Starkway` contract inherits OpenZeppelin's `Ownable` contract, which allows for managing owner addresses with a one-step change mechanism. On the other side, the L2 part of the protocol utilizes `admin_auth`, which allows for much more secure admin management - addition or removal of admin must be done by two separate currently registered admins. This mechanism is much more secure and narrows the possibility of mistakenly changing the owner to an incorrect address or malicious action by a single admin caused by, e.g., losing a private key.

Recommendation(s): Consider increasing the security of the L1 owner mechanism that would not rely on a single and directly transferred owner. Increased security may be achieved by using `Ownable2Step` that allows two-step owner changing, or use `AccessControl` which can be modified by to be similar to current L2 admin management mechanism.

Status: Fixed

Update from the client: Both Starkway and Starkway now inherit from Ownable2Step. Ownable2Step is available only in the latest major version of OZ contracts and requires updating the versions of Solidity and dependent packages. Hence we decided to import the necessary OZ contracts directly in our project. The changes can be found in ec481f7.

6.19 [Info] Zero address check exists only for L1 ETH transfers

File(s): `helpers/TokenUtils.sol`

Description: The transfers of funds should not happen to the zero address unless the funds are meant to be burned, which is usually done with an explicit burn function. The L1 `Starkway` protocol protocol checks for the `0` address for ETH transfers in the function `transferFundsTo(...)`. However, this check is not applied for ERC20 token transfers, as presented below:

```
function transferFundsTo(address token, address to, uint256 amount) internal {
  if (token == ETH_ADDRESS) {
    // @audit-note zero address check only made for the ETH transfers
    require(to != address(0));
    (bool isSuccess,) = to.call{value: amount}("");
    if (!isSuccess) {
      revert TokenUtils__EthTransferFailed({
        to: to,
        value: amount
      });
    }
  } else {
    // @audit-note no check for zero address for ERC20 tokens
    IERC20(token).safeTransfer(to, amount);
  }
}
```

Moreover, the zero address checks are not done on the L2 `starkway` contract during the funds' withdrawal. Lack of this check may result in sending a message to L1 `Starkway` contract with the `0` address.

Recommendation(s): Consider checking if the recipient is not the `0` address consistently on both L1 and L2 contracts and for all tokens, including ERC20.

Status: Fixed

Update from the client: A proposed check was added on the L2 side in 5930422. On the L1 side the issue was addressed in 421d82b and the related tests were added in 6220b53.

6.20 [Best Practice] Unnecessary traits are derived for custom types

File(s): `datatypes.cairo`

Description: The custom types in `datatypes` derive both `Drop` and `Destruct` traits. Those traits are equivalent, with two differences: `Drop` can't be derived by types with dictionaries, and is `no-op` meaning it does not generate CASM. Therefore the types in `datatypes` do not need to derive both the `Destruct` and `Drop` traits.

Recommendation(s): Consider deriving the correct traits for the custom types.

Status: Fixed

Update from the client: Addressed in 682ede0.

6.21 [Best Practice] Using different versions of Cairo syntax

File(s): `datatypes.cairo`

Description: The implementations `TokenAmountPartialOrd` and a `TokenAmountPartialEq` are using different syntax:

```
// TokenAmountPartialOrd
#[inline_always]
// TokenAmountPartialEq
#[inline(always)]
```

Recommendation(s): Unify the syntax to match one Cairo version across the contracts.

Status: Fixed

Update from the client: Addressed in d71ca28.

6.22 [Best Practices] Check bridge interface before writing to storage.

File(s): `Starkway.cairo`

Description: Registering the bridge through the `register_bridge_adapter(...)` method checks that the bridge registered before with the same ID. It makes it impossible to change the address in the future. However, bridge adapters that don't fit the current interface can be added through this method.

Recommendation(s): Consider checking that the contract address is a valid bridge adapter.

Status: Fixed

Update from the client: Addressed in 7db387e.

6.23 [Best Practices] Unnecessary storage mapping `everConnectedAsStarknet`

File(s): `PairedToL2.sol`

Description: Storage mapping named `everConnectedAsStarknet` is unnecessary. It is just written once in the following lines:

```
function _setStarknetAddressTo(address newAddress) internal {
  // ...
  everConnectedAsStarknet[newAddress] = true;
  // ...
}
```

But never read from somewhere else. Also, it is an internal variable that any external view calls can't reach this mapping.

Recommendation(s): Consider removing this variable and remove assigning at `_setStarknetAddressTo(...)` method.

Status: Fixed

Update from the client: The unused storage variable was removed in bc635b4.

6.24 [Best Practices] Unnecessary usage of storage slot

File(s): `fee_library.cairo`

Description: On FeeLibrary, `default_fee_rate` stored in u256 which stores into 2 slots of felt252. However, the value can be settled only in `set_default_fee_rate(...)`, and there is a check for the new value that can not be higher than 300.

```
#[storage]
struct Storage {
    default_fee_rate: u256, // @audit-issue: Unnecessary store of u256. Value can fit in one slot.
    // ...
}
```

```
fn set_default_fee_rate(ref self: ContractState, default_fee_rate: u256) {
    // @audit-issue: default_fee_rate can fit in one felt252 slot. It's better to store it in one slot.
    let MAX_FEE_RATE = u256 { low: 300, high: 0 };
    assert(default_fee_rate <= MAX_FEE_RATE, 'Default_fee_rate > MAX_FEE_RATE');
    self.default_fee_rate.write(default_fee_rate);
}
```

Recommendation(s): Consider using unsigned integer types that fit into one felt slot.

Status: Fixed

Update from the client: Addressed in 98515e3. `u16` data type now used instead of `u256` data type.

6.25 [Best Practices] Use `BoundedInt` to represent unsigned integer max values

File(s): `erc20.cairo`

Description: The function `_spend_allowance(...)` will only reduce a spenders allowance if they do not already have an unlimited allowance for a given owner address. This unlimited allowance is determined by generating a bitmask representing the maximum possible value for a `u128` type and comparing against both the high and low components of the `u256` type, as shown below:

```
let ONES_MASK = 0xffffffffffffffffffffffffffffffff_u128;
let is_unlimited_allowance = ((current_allowance.low == ONES_MASK)
                            & (current_allowance.high == ONES_MASK));
```

Recommendation: The function `BoundedInt::max()` from the Cairo core library can be used instead, as shown below:

```
let is_unlimited_allowance = current_allowance == integer::BoundedInt::max();
```

Status: Fixed

Update from the client: Addressed in d050717.

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. **The documentation of the ZKX Starkway is provided in extensive inline comments over the code**.

8 Test Suite Evaluation

8.1 Solidity Contracts Compilation

```
> npx hardhat compile
Generating typings for: 33 artifacts in dir: typechain-types for target: ethers-v5
Successfully generated 96 typings!
Compiled 34 Solidity files successfully
```

8.2 Cairo Contracts Compilation

```
> scarb build
   Compiling starkway v0.1.0 (~/NM-0120-Security-Review-ZKX/L2/Scarb.toml)
    Finished release target(s) in 11 seconds
```

8.3 Solidity Tests Output

```
npx hardhat test
Downloading compiler 0.8.17
Compiled 34 Solidity files successfully


  Deposit cancelation
     Cancel ERC20 deposit by User
     Cancel ERC20 deposit by Admin
     Cancel ERC20 deposit with message by User
     Cancel ERC20 deposit with message by Admin

  Extract Deposit nonce
     Nonce from Deposit event
     Nonce from typed events query
     Nonce from raw logs query

  ERC20 Deposits
     Revert Deposit if amount == 0
     Revert Deposit if L2 address == 0x00
     Revert Deposit if L2 address is invalid
     Revert Deposit if amount < MIN deposit
     Revert Deposit if amount > MAX deposit
     Revert Deposit with message if message recipient address is 0x00
     Success Deposit when token is not yet initialized
     Success Deposit when token is already initialized
     Success Deposit with message

  ETH Deposits
     Revert Deposit if amount == 0
     Revert Deposit if L2 address == 0x00
     Revert Deposit if L2 address is invalid
     Revert Deposit if amount < MIN deposit
     Revert Deposit if amount > MAX deposit
     Revert Deposit with message if message recipient address is 0x00
     Successful Deposit when ETH is not yet initialized
     Successful Deposit when ETH is already initialized
     Success Deposit with message

  Token Settings Errors
     Revert if token not initialized
     Revert if useCustomFeeRate == true, but no segments
     Revert if useCustomFeeRate == false, but with segments
     Revert if min fee > max fee
     Revert if min deposit > max deposit
     Revert if min fee > min deposit
     Revert if segments not in increasing order
     Revert if any segments fee rate > MAX_FEE_RATE
     Revert if segments don't cover full deposit range
     Revert if not-last segment's to-amount == 0
```

```
Token Settings Setup
   Emits event
   Clear settings
   Set 0-to-0 fee range
   Set A-to-0 fee range
   Set 0-to-B fee range
   Set A-to-B fee range
   Set 0-to-0 deposit range
   Set A-to-0 deposit range
   Set 0-to-B deposit range
   Set A-to-B deposit range
   Set single fee segment with 0 (unlimited) to-amount
   Set single fee segment with specific to-amount
   Set several segments, last one is limited
   Set several segments, last one is unlimited

Starkway deployment
   Revert when Starknet address is ZERO
   Revert when vault address is ZERO
   Revert when Starkway L2 address is ZERO
   Revert when fee rate too high
   Success deployment

StarkwayHelper with Malicious Tokens
   Resistant to revert of IERC20.balanceOf() call
   Resistant to revert of IERC20.name() call
   Resistant to revert of IERC20.symbol() call
   Resistant to revert of IERC20.decimals() call
   Resistant to revert of all IERC20Metadata calls
   Resistant to 100 infinite loop tokens

StarkwayHelper with MANY tokens
   Return MANY tokens (Full)
   Return MANY tokens (Short)
   Return MANY tokens with Multiplier (Full)
   Return MANY tokens with Multiplier (Short)
   Behaviour of `skipZeroBalances` setting
   Return 0 tokens

StarkwayHelper with ONE token
   Return 1 token

Token/ETH initialization
   Init call during token deposit
   Init call during ETH deposit
   Standalone token init (by admin)
   Standalone token init (by random user)
   Standalone ETH init (by admin)
   Standalone ETH init by some user
   NOT possible to init Token twice
   NOT possible to init ETH twice

ERC20 Withdrawals
   Revert ERC20 withdrawal when token is not yet initialized
   Successful ERC20 withdrawal by user
   Successful ERC20 withdrawal by admin

ETH Withdrawals
   Revert ETH withdrawal when token is not yet initialized
   Successful ETH withdrawal by user
   Successful ETH withdrawal by admin

81 passing (1m)
```

8.4 Cairo Tests Output

```
> cairo-test --starknet .
running 160 tests
test tests::test_erc20::test_erc20::test_transfer_from_to_zero_address ... ok
test tests::test_admin_auth::test_admin_auth::test_constructor ... ok
test tests::test_erc20::test_erc20::test_transfer_from_from_zero_address ... ok
test tests::test_starkway::test_starkway::test_setting_reentrancy_guard_classhash_with_unauthorized_user ... ok
test tests::test_withdraw_multi::test_withdraw_multi::test_withdraw_uninitialized_token ... ok
test tests::test_starkway::test_starkway::test_setting_l1_starkway_address_with_unauthorized_user ... ok
test tests::test_authorised_init_token::test_authorised_init_token::test_init_with_zero_token_name ... ok
test tests::test_starkway::test_starkway::test_setting_admin_auth_address_with_authorized_user ... ok
test tests::test_admin_auth::test_admin_auth::test_set_min_number_admins ... ok
test tests::test_fee_library::test_fee_library::test_set_fee_range_unauthorized ... ok
test tests::test_admin_auth::test_admin_auth::test_remove_admin ... ok
test tests::test_erc20::test_erc20::test_increase_allowance ... ok
test tests::test_can_withdraw_multi::test_can_withdraw_multi::test_out_of_range ... ok
test tests::test_prep_withdrawal::test_prep_withdrawal::test_no_token ... ok
test tests::test_admin_auth::test_admin_auth::test_adds_zero_address_as_admin ... ok
test tests::test_erc20::test_erc20::test_increase_allowance_to_zero_address ... ok
test tests::test_admin_auth::test_admin_auth::test_remove_admin_who_is_not_admin ... ok
test tests::test_authorised_init_token::test_authorised_init_token::test_init_with_zero_token_symbol ... ok
test tests::test_starkway::test_starkway::test_setting_reentrancy_guard_classhash_with_authorized_user ... ok
test tests::test_starkway::test_starkway::test_setting_l1_starkway_address_with_authorized_user ... ok
test tests::test_starkway::test_starkway::test_setting_erc20_classhash_with_unauthorized_user ... ok
test tests::test_erc20::test_erc20::test_increase_allowance_from_zero_address ... ok
test tests::test_admin_auth::test_admin_auth::test_non_admin_removes_or_adds_admin ... ok
test tests::test_admin_auth::test_admin_auth::test_add_admin ... ok
test tests::test_withdraw_multi::test_withdraw_multi::test_zero_amount ... ok
test tests::test_starkway_withdraw::test_starkway_withdraw::test_withdraw_non_whitelisted_token ... ok
test tests::test_fee_library::test_fee_library::test_set_and_get_fee_range ... ok
test tests::test_erc20::test_erc20::test_decrease_allowance ... ok
test tests::test_starkway::test_starkway::test_register_bridge_adapter_with_unauthorized_user ... ok
test tests::test_authorised_init_token::test_authorised_init_token::test_init_with_invalid_decimal_range ... ok
test tests::test_starkway::test_starkway::test_setting_l1_starkway_vault_address_with_unauthorized_user ... ok
test tests::test_admin_auth::test_admin_auth::test_add_admin_with_same_approvers ... ok
test tests::test_starkway::test_starkway::test_setting_erc20_classhash_with_authorized_user ... ok
test tests::test_erc20::test_erc20::test_decrease_allowance_to_zero_address ... ok
test tests::test_authorised_init_token::test_authorised_init_token::test_init_with_unauthorized_user ... ok
test tests::test_prep_withdrawal::test_prep_withdrawal::test_native_only_insufficient ... ok
test tests::test_erc20::test_erc20::test_decrease_allowance_from_zero_address ... ok
test tests::test_starkway::test_starkway::test_register_bridge_adapter_with_zero_id ... ok
test tests::test_withdraw_multi::test_withdraw_multi::test_out_of_range ... ok
test tests::test_starkway::test_starkway::test_setting_l1_starkway_vault_address_with_authorized_user ... ok
test tests::test_fee_library::test_fee_library::test_set_fee_range_with_min_greater_than_max ... ok
test tests::test_starkway::test_starkway::test_setting_fee_lib_classhash_with_unauthorized_user ... ok
test tests::test_erc20::test_erc20::test__mint ... ok
test tests::test_authorised_init_token::test_authorised_init_token::test_init_with_zero_erc20_class_hash ... ok
test tests::test_can_withdraw_multi::test_can_withdraw_multi::test_l1_address_mismatch ... ok
test tests::test_erc20::test_erc20::test__mint_to_zero ... ok
test tests::test_starkway::test_starkway::test_register_bridge_adapter_with_zero_address ... ok
test tests::test_starkway::test_starkway::test_setting_admin_auth_address_with_unauthorized_user ... ok
test tests::test_fee_library::test_fee_library::test_set_fee_segment_unauthorized ... ok
test tests::test_starkway::test_starkway::test_setting_fee_lib_classhash_with_authorized_user ... ok
test tests::test_authorised_init_token::test_authorised_init_token::test_init_token ... ok
test tests::test_erc20::test_erc20::test_burn ... ok
test tests::test_erc20::test_erc20::test_approve ... ok
test tests::test_withdraw_multi::test_withdraw_multi::test_fee_mismatch ... ok
test tests::test_starkway::test_starkway::test_register_bridge_adapter_with_invalid_name ... ok
test tests::test_erc20::test_erc20::test_burn_from_zero ... ok
test tests::test_erc20::test_erc20::test_approve_from_zero ... ok
test tests::test_prep_withdrawal::test_prep_withdrawal::test_non_native_only_insufficient ... ok
test tests::test_starkway_helper::test_starkway_helper::test_get_supported_tokens ... ok
test tests::test_erc20::test_erc20::test_approve_to_zero ... ok
test tests::test_erc20::test_erc20::test_transfer_ownership ... ok
test tests::test_starkway::test_starkway::test_setting_withdrawal_range_with_unauthorized_user ... ok
test tests::test_fee_library::test_fee_library::test_set_fee_segment_with_invalid_tier ... ok
test tests::test_erc20::test_erc20::test_transfer_ownership_revert ... ok
test tests::test_starkway::test_starkway::test_registering_already_existing_bridge ... ok
test tests::test_starkway_withdraw::test_starkway_withdraw::test_non_native_withdrawal ... ok
test tests::test_authorised_init_token::test_authorised_init_token::test_initialising_already_initialised_token ... ok
test tests::test_utils::test_utils::test_sort_basic ... ok
```

```
test tests::test_erc20::test_erc20::test_transfer ... ok
test tests::test_utils::test_utils::test_sort_token_amounts ... ok
test tests::test_utils::test_utils::test_reverse_array ... ok
test tests::test_starkway::test_starkway::test_setting_withdrawal_range_for_unregistered_token ... ok
test tests::test_withdraw_multi::test_withdraw_multi::test_incompatible_l1_address ... ok
test tests::test_erc20::test_erc20::test_transfer_not_enough_balance ... ok
test tests::test_can_withdraw_multi::test_can_withdraw_multi::test_insufficient_single_token_liquidity ... ok
test tests::test_prep_withdrawal::test_prep_withdrawal::test_non_native_only_sufficient ... ok
test tests::test_fee_library::test_fee_library::test_set_fee_segment_with_invalid_tier1_amount ... ok
test tests::test_can_withdraw_multi::test_can_withdraw_multi::test_uninitialized_token ... ok
test tests::test_erc20::test_erc20::test_transfer_from_zero ... ok
test tests::test_fee_library::test_fee_library::test_set_fee_lib_class_hash_unauthorized ... ok
test tests::test_fee_library::test_fee_library::test_set_default_fee_rate_unauthorized ... ok
test tests::test_whitelist_token::test_whitelist_token::test_whitelist_token_with_unauthorized_user ... ok
test tests::test_erc20::test_erc20::test_transfer_to_zero ... ok
test tests::test_prep_withdrawal::test_prep_withdrawal::test_mixed_insufficient ... ok
test tests::test_can_withdraw_multi::test_can_withdraw_multi::test_single_native_token_sufficient_liquidity ... ok
test tests::test_starkway::test_starkway::test_setting_withdrawal_range_for_registered_token ... ok
test tests::test_erc20::test_erc20::test_transfer_from ... ok
test tests::test_can_withdraw_multi::test_can_withdraw_multi::test_empty_transfer_list ... ok
test tests::test_fee_library::test_fee_library::test_set_fee_segment_with_invalid_higher_fee ... ok
test tests::test_whitelist_token::test_whitelist_token::test_whitelist_non_existing_bridge_adapter ... ok
test tests::test_fee_library::test_fee_library::test_set_and_get_default_fee_rate ... ok
test tests::test_erc20::test_erc20::test_transfer_from_doesnt_consume_infinite_allowance ... ok
test tests::test_erc20::test_erc20::test_transfer_from_greater_than_allowance ... ok
test tests::test_prep_withdrawal::test_prep_withdrawal::test_mixed_2_sufficient ... ok
test tests::test_can_withdraw_multi::test_can_withdraw_multi::test_single_native_token_sufficient_prior_liquidity ...
↪  ok
test tests::test_whitelist_token::test_whitelist_token::test_whitelist_zero_address_token ... ok
test tests::test_fee_library::test_fee_library::test_get_fee_rate_without_setting_fee_segment ... ok
test tests::test_starkway_helper::test_starkway_helper::test_get_non_native_token_balances ... ok
test tests::test_prep_withdrawal::test_prep_withdrawal::test_native_only_sufficient ... ok
test tests::test_withdraw_multi::test_withdraw_multi::test_l1_address_mismatch ... ok
test tests::test_can_withdraw_multi::test_can_withdraw_multi::test_amount_zero ... ok
test tests::test_prep_withdrawal::test_prep_withdrawal::test_mixed_2_more_than_sufficient ... ok
test tests::test_fee_library::test_fee_library::test_set_fee_segment_with_invalid_higher_amount ... ok
test tests::test_whitelist_token::test_whitelist_token::test_whitelist_zero_address_bridge_adapter ... ok
test tests::test_fee_library::test_fee_library::test_get_fee_rate_after_setting_single_fee_segment ... ok
test tests::test_can_withdraw_multi::test_can_withdraw_multi::test_incompatible_l1_address ... ok
test tests::test_withdraw_multi::test_withdraw_multi::test_no_approval ... ok
test tests::test_withdraw_multi::test_withdraw_multi::test_amount_mismatch ... ok
test tests::test_can_withdraw_multi::test_can_withdraw_multi::test_single_non_native_token_sufficient_liquidity ... ok
test tests::test_whitelist_token::test_whitelist_token::test_whitelist_uninitialized_token ... ok
test tests::test_whitelist_token::test_whitelist_token::test_whitelist_token ... ok
test tests::test_withdraw_multi::test_withdraw_multi::test_sufficient_single_non_native ... ok
test tests::test_prep_withdrawal::test_prep_withdrawal::test_different_approvals ... ok
test tests::test_fee_library::test_fee_library::test_set_fee_segment_with_invalid_lower_fee ... ok
test tests::test_prep_withdrawal::test_prep_withdrawal::test_different_approvals_2 ... ok
test tests::test_can_withdraw_multi::test_can_withdraw_multi::test_amount_mismatch ... ok
test tests::test_fee_library::test_fee_library::test_get_fee_rate_after_setting_multiple_fee_segments ... ok
test tests::test_withdraw_multi::test_withdraw_multi::test_insufficient_user_balance ... ok
test tests::test_starkway_withdraw::test_starkway_withdraw::test_greater_than_withdrawal_range ... ok
test tests::test_withdraw_admin_fees::test_withdraw_admin_fees::test_withdraw_with_zero_withdrawal_amount ... ok
test tests::test_whitelist_token::test_whitelist_token::test_whitelist_multiple_tokens ... ok
test tests::test_erc20::test_erc20::test_constructor ... ok
test tests::test_can_withdraw_multi::test_can_withdraw_multi::test_multi_tokens_sufficient_prior_liquidity ... ok
test tests::test_fee_library::test_fee_library::test_set_fee_segment_with_invalid_lower_amount ... ok
test tests::test_withdraw_multi::test_withdraw_multi::test_sufficient_single_native ... ok
test tests::test_withdraw_admin_fees::test_withdraw_admin_fees::test_withdraw_with_unauthorized_user ... ok
test tests::test_can_withdraw_multi::test_can_withdraw_multi::test_multi_tokens_sufficient_prior_liquidity_2 ... ok
test tests::test_withdraw_admin_fees::test_withdraw_admin_fees::test_withdraw_with_non_whitelisted_token ... ok
test tests::test_withdraw_multi::test_withdraw_multi::test_insufficient_multi_token ... ok
test tests::test_can_withdraw_multi::test_can_withdraw_multi::test_multi_tokens_sufficient_liquidity ... ok
test tests::test_starkway_withdraw::test_starkway_withdraw::test_lesser_than_withdrawal_range ... ok
test tests::test_withdraw_admin_fees::test_withdraw_admin_fees::test_withdraw_with_uninitialised_token ... ok
test tests::test_starkway_helper::test_starkway_helper::test_get_supported_tokens_with_balance ... ok
test tests::test_fee_library::test_fee_library::test_set_fee_segment_with_invalid_tier_number ... ok
test tests::test_starkway_withdraw::test_starkway_withdraw::test_withdraw_uninitialized_token ... ok
test tests::test_prep_withdrawal::test_prep_withdrawal::test_uninitialized_token ... ok
test tests::test_withdraw_multi::test_withdraw_multi::test_sufficient_multi_token_4 ... ok
test tests::test_fee_library::test_fee_library::test_calculate_fee_without_setting_fee_segment ... ok
test tests::test_withdraw_admin_fees::test_withdraw_admin_fees::test_withdraw_with_zero_address_recipient ... ok
test tests::test_fee_library::test_fee_library::test_calculate_fee_after_setting_fee_segment ... ok
test tests::test_starkway_withdraw::test_starkway_withdraw::test_fee_mismatch ... ok
```

```
test tests::test_fee_library::test_fee_library::test_calculate_fee_with_fee_less_than_fee_range_min ... ok
test tests::test_withdraw_admin_fees::test_withdraw_admin_fees::test_withdrawal_amount_exceeds_fee ... ok
test tests::test_prep_withdrawal::test_prep_withdrawal::test_out_of_range ... ok
test tests::test_upgradeability::test_upgradeability::test_invalid_call_without_upgrade ... ok
test tests::test_starkway_withdraw::test_starkway_withdraw::test_mint_and_withdraw ... ok
test tests::test_withdraw_multi::test_withdraw_multi::test_sufficient_multi_token_1 ... ok
test tests::test_upgradeability::test_upgradeability::test_unauthorised_upgrade ... ok
test tests::test_fee_library::test_fee_library::test_calculate_fee_with_is_set_flag_disabled ... ok
test tests::test_fee_library::test_fee_library::test_calculate_fee_with_fee_more_than_fee_range_max ... ok
test tests::test_fee_library::test_fee_library::test_calculate_fee_after_setting_fee_segment_and_fee_range ... ok
test tests::test_prep_withdrawal::test_prep_withdrawal::test_zero_amount ... ok
test tests::test_upgradeability::test_upgradeability::test_zero_class_hash ... ok
test tests::test_starkway_withdraw::test_starkway_withdraw::test_without_approval ... ok
test tests::test_starkway_withdraw::test_starkway_withdraw::test_insufficient_balance ... ok
test tests::test_upgradeability::test_upgradeability::test_simple_upgrade ... ok
test tests::test_withdraw_multi::test_withdraw_multi::test_sufficient_multi_token_2 ... ok
test tests::test_withdraw_admin_fees::test_withdraw_admin_fees::test_withdraw_with_native_token ... ok
test tests::test_withdraw_admin_fees::test_withdraw_admin_fees::test_withdraw_with_non_native_token ... ok
test tests::test_withdraw_multi::test_withdraw_multi::test_sufficient_multi_token_3 ... ok
test result: ok. 160 passed; 0 failed; 0 ignored; 0 filtered out;
```

8.5 Slither

All the relevant issues raised by Slither have been incorporated into the issues described in this report.

9 About Nethermind

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

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

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

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

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

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

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

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

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

Learn more about us at nethermind.io.

General Advisory to Clients

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

Disclaimer

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