Security Review Report
NM-0544B Token Bridge

NETHERMIND
SECURITY

(July 1, 2025)

Contents

1 Executive Summary

This document presents the results of a security review conducted by Nethermind Security for a system of smart contracts referred to as the **Token Bridge** developed by Karnot. The audited contracts are designed to facilitate the secure transfer of assets between the Starknet Layer 2 (L2) network and bespoke Layer 3 (L3) networks, known as Appchains.

This bridge is a crucial infrastructure component within the **Piltover** ecosystem, a system developed to extend Starknet by enabling the creation and operation of L3 solutions. By providing a reliable mechanism for liquidity to flow from L2 to L3, the Token Bridge is a foundational component for Appchains built with frameworks like Madara.

The bridge allows users to lock tokens in the L2 contract, which then uses the Piltover messaging protocol to communicate with a corresponding contract on an L3 Appchain to mint an equivalent amount of wrapped tokens. The reverse process, where tokens are burned on L3, initiates a message to unlock the original assets on L2. The architectural design is inspired by the official Starkgate L1-L2 bridge, adapting its proven concepts for the unique requirements of the L2-L3 environment.

The audit comprises 1964 lines of Cairo code. The audit was performed using (a) manual analysis of the codebase, (b) automated analysis tools, and (c) creation of test cases.

Along this document, we report 9 points of attention, where one is classified as `Low`, and eight are classified as `Info` and `Best Practices` severity. The issues are summarized in Fig. 1.

This document is organized as follows. Section 2 presents the files in the scope. Section 3 summarizes the issues. Section 4 presents the system overview. Section 5 discusses the risk rating methodology. Section 6 details the issues. Section 7 discusses the documentation provided by the client for this audit. Section 8 presents the test output. Section 9 concludes the document.

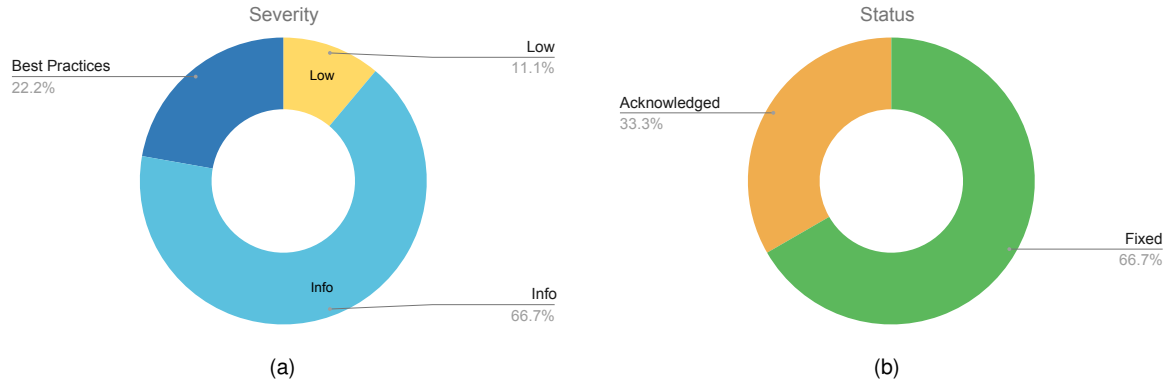

Fig. 1: Distribution of issues: Critical (0), **High** (0), **Medium** (0), **Low** (1), **Undetermined** (0), **Informational** (6), **Best Practices** (2). **Distribution of status: Fixed** (6), **Acknowledged** (3), **Mitigated** (0), **Unresolved** (0)

Summary of the Audit

Audit Type	Security Review
Initial Report	June 23, 2025
Final Report	July 1, 2025
Repository	karnotxyz/starknet_bridge
Initial Commit	65a441c9519261661e85714f01363f49a89bbb35
Final Commit	eb39c9ecd50aa8cbbe75bb0df205821c2a8f68bc
Documentation	README
Documentation Assessment	High
Test Suite Assessment	High

2 Audited Files

	Contract	LoC	Comments	Ratio	Blank	Total
1	src/constants.cairo	15	10	66.7%	10	35
2	src/lib.cairo	57	9	15.8%	15	81
3	src/withdrawal_limit/component.cairo	125	31	24.8%	19	175
4	src/withdrawal_limit/interface.cairo	6	1	16.7%	1	8
5	src/access_control/component.cairo	151	9	6.0%	27	187
6	src/access_control/roles.cairo	8	8	100.0%	1	17
7	src/timelock/timelock.cairo	77	16	20.8%	17	110
8	src/erc20/roles_interface.cairo	115	36	31.3%	26	177
9	src/erc20/err_msg.cairo	34	0	0.0%	2	36
10	src/erc20/interface.cairo	9	2	22.2%	2	13
11	src/erc20/erc20.cairo	471	158	33.5%	83	712
12	src/erc20/replaceability_interface.cairo	37	17	45.9%	13	67
13	src/erc20/access_control_interface.cairo	21	15	71.4%	6	42
14	src/bridge/types.cairo	20	3	15.0%	5	28
15	src/bridge/interface.cairo	80	2	2.5%	12	94
16	src/bridge/token_bridge.cairo	738	181	24.5%	138	1057
	Total	**1964**	**498**	**25.4%**	**377**	**2839**

3 Summary of Issues

	Finding	Severity	Update
1	Missing token balance checks on `ERC20` transfer functions	Low	Fixed
2	Inconsistent ERC20 Standard Usage	Info	Acknowledged
3	Incorrect max balance exceeded assertion in the `accept_deposit(...)` function	Info	Fixed
4	System does not support ERC20 tokens that lack the optional `decimals` function	Info	Acknowledged
5	Token activation can fail due to a race condition	Info	Fixed
6	Token settings are not cleared upon unsuccessful token activation	Info	Fixed
7	Tokens in the pending state cannot be blocked or deactivated	Info	Acknowledged
8	Incorrect error message in the `decrease_withdrawal_limit(...)` function	Best Practices	Fixed
9	Missing event emission during unsuccessful token activation	Best Practices	Fixed

4 System Overview

The Token Bridge is a set of smart contracts designed to facilitate the transfer of assets between the Starknet Layer 2 (L2) network and a Starknet-based application-specific chain (Appchain, or L3). The primary objective of the bridge is to provide a secure and reliable mechanism for users to lock tokens on L2 and receive corresponding wrapped tokens on the L3 Appchain, and vice-versa.

The architecture and design of the Token Bridge are heavily inspired by the official Starknet (Starkgate) bridge, which bridges assets between Ethereum (L1) and Starknet (L2). The reference implementation, specifically the StarknetTokenBridge.sol contract, is written in Solidity and deployed on Ethereum.

In contrast, the Token Bridge contracts are intended for deployment on Starknet (L2). This L2 bridge communicates with a corresponding Cairo contract on the L3 Appchain via the Piltover messaging protocol, which is the Cairo implementation of Starknet's core messaging contracts.

4.1 Architecture and Core Components

The system is centered around the `token_bridge.cairo` contract, which manages all core bridging functionalities on Starknet L2.

Key components and features include:

- **Token Management**: The bridge implements a comprehensive token lifecycle management system. Tokens can be enrolled, activated, deactivated, or blocked. The status of a token is tracked via the `TokenStatus` enum, which can be one of `Unknown`, `Pending`, `Active`, `Deactivated`, or `Blocked`.
- **Bridging Operations**:
  - `deposit(...)` & `deposit_with_message(...)`: Users lock ERC20 tokens on L2 by calling these functions. This action sends a message to the L3 Appchain to mint an equivalent amount of the corresponding token.
  - `withdraw(...)`: To withdraw assets, a user first burns the tokens on L3, which sends a message back to the L2 bridge. The user then calls `withdraw(...)` on the L2 contract to consume this message and unlock the original tokens.
  - **Message Cancellation:** The bridge includes functions like `deposit_cancel_request(...)` and `deposit_reclaim(...)` to allow users to reclaim their assets if a deposit message fails to be consumed on L3 within a specified time frame.
- **Security Mechanisms**:
  - **Total Balance Limits**: The `APP_GOVERNOR` can set a `max_total_balance` for each token. This acts as a global deposit limit, preventing the total value locked (TVL) for a specific token from exceeding a predefined cap, thereby controlling the overall risk exposure of the bridge.
  - **Withdrawal Limits**: A `WithdrawalLimitComponent` provides functionality to enforce daily withdrawal quotas on a per-token basis, mitigating the potential impact of a security incident.
  - **Pausable Contract**: The contract can be paused by a designated `SECURITY_AGENT` role in case of an emergency, halting all major operations like deposits and withdrawals.
  - **Reentrancy Guard**: Utilizes OpenZeppelin's `ReentrancyGuardComponent` to prevent reentrancy attacks on state-changing functions.

4.2 Access Control

The Token Bridge implements a granular, hierarchical role-based access control (RBAC) system to manage administrative and security-sensitive functions. This system is more elaborate than the one found in the reference Starkgate implementation. The roles and their administrative hierarchy are defined as follows:

Role	Role Admin
UPGRADE_GOVERNOR	DEFAULT_ADMIN (Timelock Contract)
GOVERNANCE_ADMIN	GOVERNANCE_ADMIN (Self-managed)
APP_GOVERNOR	GOVERNANCE_ADMIN
SECURITY_ADMIN	GOVERNANCE_ADMIN
SECURITY_AGENT	SECURITY_ADMIN
TOKEN_ADMIN	APP_GOVERNOR

Table 2: Karnot Bridge Role Hierarchy

A key security feature is the integration of a `TimelockController` contract. The `UPGRADE_GOVERNOR` role, which has the sole authority to upgrade the bridge contract, is assigned exclusively to the Timelock contract. This ensures that any contract upgrade is subject to a mandatory time delay, providing an opportunity for users and guardians to review the proposed changes.

4.3 Key Differences from Starkgate (L1) Implementation

While based on the Starkgate bridge, the Token Bridge introduces several notable architectural and functional differences.

- **Operating Layers & Language**: The audited contract operates on L2 (Starknet) and is written in Cairo, communicating with an L3 Appchain. The reference Starkgate bridge operates on L1 (Ethereum), is written in Solidity, and communicates with L2 (Starknet).
- **Token Enrollment**: Starkgate requires a 'manager' role to enroll new tokens. The Token Bridge introduces a configurable `permissioned_enroll` flag. When disabled (the default), token enrollment is permissionless. When enabled by the `APP_GOVERNOR`, enrollment is restricted to the `TOKEN_ADMIN` role.
- **Token Status Management**: Token Bridge extends the token lifecycle with more granular states.
- **Withdrawal Limits**: In Starkgate's `WithdrawalLimit.sol`, the withdrawal limit is a single percentage applied globally to all tokens. Token Bridge's `WithdrawalLimitComponent` improves upon this by allowing for per-token daily withdrawal percentage limits. Furthermore, the ability to modify these limits is split: `SECURITY_ADMIN` can only increase or disable limits, while the lower-privileged `SECURITY_AGENT` can only decrease them, creating a safer operational hierarchy.

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 [Low] Missing token balance checks on ERC20 transfer functions

File(s): src/bridge/token_bridge.cairo

Description: The token_bridge contract allows users to withdraw tokens through several functions: withdraw(...), deposit_with_message_reclaim(...), and deposit_reclaim(...). In all these cases, the function first consumes a message to authorize the transfer and then proceeds to send the tokens to the recipient.

The issue is that these functions do not verify whether the token transfer was successful. The message is consumed optimistically before the transfer(...) call is made. Some ERC20 tokens may not revert on a failed transfer (e.g., if the token is paused or the recipient is blacklisted) but instead return a boolean status. Since this return value is not checked, the transaction can succeed even if the tokens were not actually moved.

This can lead to a scenario where a user's withdrawal message is consumed, but they never receive their tokens due to a failed transfer. Because the message is marked as used, the user cannot retry the operation, resulting in their funds being permanently stuck in the bridge contract. It is worth noting that the accept_deposit(...) function correctly checks for transfer success by comparing balances before and after the call, but this pattern was not applied to the withdrawal and reclaim flows.

For brevity, only the withdraw(...) function is shown, as the reclaim functions follow the same flawed logic.

```
fn withdraw(
    ref self: ContractState,
    token: ContractAddress,
    amount: u256,
    recipient: ContractAddress,
) {
    // ...
    // @audit The message is consumed before the transfer is attempted.
    self.consume_message(token, amount, recipient);

    let tokenDispatcher = IERC20Dispatcher { contract_address: token };

    // @audit-issue The success of this transfer is not verified.
    // If it fails, the message is already consumed and funds are lost.
    tokenDispatcher.transfer(recipient, amount);
    // ...
}
```

Recommendation(s): Consider validating that the token transfer was successful in the withdraw(...), deposit_with_message_reclaim(...), and deposit_reclaim(...) functions. This can be achieved similarly to the implementation in the accept_deposit(...) function.

Status: Fixed

Update from the client: Fixed in 82611ee.

6.2 [Info] Inconsistent ERC20 Standard Usage

File(s): src/bridge/token_bridge.cairo

Description: The `accept_deposit(...)` function is responsible for handling user deposits of ERC20 tokens into the contract. It verifies that the token is supported, checks against a maximum balance, and then performs a `transfer_from(...)` operation to move tokens from the caller to the contract's address. Finally, it asserts that the transfer was successful by comparing the contract's balance before and after the transfer.

The issue lies in the fact that Starknet ERC20 tokens can have different function signatures, specifically `camelCase` (e.g., `transferFrom(...)`, `balanceOf(...)`) or `snake_case` (e.g., `transfer_from(...)`, `balance_of(...)`). While the latest standard recommends using `snake_case`, older tokens might still adhere to `camelCase`. The current implementation exclusively uses `snake_case` for `balance_of(...)` and `transfer_from(...)` calls.

```
fn accept_deposit(self: @ContractState, token: ContractAddress, amount: u256) {
    // ...
    let dispatcher = IERC20Dispatcher { contract_address: token };

    // @audit Using snake_case
    let current_balance: u256 = dispatcher.balance_of(get_contract_address());
    let max_total_balance = self.get_max_total_balance(token);
    assert(current_balance + amount < max_total_balance, Errors::MAX_BALANCE_EXCEEDED);

    let this_address = get_contract_address();
    // @audit Using snake_case
    let initial_balance = dispatcher.balance_of(this_address);
    // @audit-issue Only snake_case `transfer_from(...)` is used.
    dispatcher.transfer_from(caller, this_address, amount);
    assert(
        // @audit Using snake_case
        dispatcher.balance_of(this_address) == initial_balance + amount,
        Errors::TOKENS_NOT_TRANSFERRED,
    );
}
```

If a user attempts to deposit an ERC20 token that uses `camelCase` function signatures, the `transfer_from(...)` call will fail. This is because the dispatcher is configured to call the `snake_case` version of the function, which does not exist on `camelCase` standard tokens.

Recommendation(s): Consider implementing a mechanism to support both `camelCase` and `snake_case` ERC20 token standards. This would ensure compatibility with a wider range of ERC20 tokens on Starknet.

Status: Acknowledged

Update from the client: We acknowledge this issue. We feel this should be fine as most of the relevant tokens on Starknet mainnet have already migrated, and going forward, the new standard says the token must have snake_case.

6.3 [Info] Incorrect max balance exceeded assertion in the `accept_deposit(...)` function

File(s): src/bridge/token_bridge.cairo

Description: In the `accept_deposit(...)` function, the contract checks if a new deposit would exceed the `max_total_balance` for a token. The check is performed using a strict less-than (<) comparison.

This prevents a deposit that would make the contract's balance exactly equal to `max_total_balance`. This behavior is inconsistent with the reference Starkgate implementation, which uses a less-than-or-equal-to (<=) check, allowing the balance to reach the maximum limit.

```
fn accept_deposit(self: @ContractState, token: ContractAddress, amount: u256) {
    // ...
    let current_balance = IERC20Dispatcher { contract_address: token }.balance_of(self.own_address);
    let max_total_balance = self.token_settings.read(token).max_total_balance;

    // @audit-issue The use of `<` prevents the total balance from ever reaching max_total_balance.
    assert(current_balance + amount < max_total_balance, Errors::MAX_BALANCE_EXCEEDED);

    // ...
}
```

Recommendation(s): Consider changing the strict less-than (<) comparison to less-than-or-equal-to (<=) to correctly allow deposits up to the maximum configured balance.

Status: Fixed

Update from the client: Fixed in ec11bc3.

6.4 [Info] System does not support ERC20 tokens that lack the optional `decimals` function

File(s): src/bridge/token_bridge.cairo

Description: The `deployment_message_payload(...)` function constructs a calldata payload containing metadata about a given token. This metadata includes the token's name, symbol, and the number of decimals. The function makes a direct external call to the `decimals()` function on the token's contract address to fetch this information.

```
pub fn deployment_message_payload(token: ContractAddress) -> Span<felt252> {
    // ...
    let dispatcher = IERC20MetadataDispatcher { contract_address: token };
    token.serialize(ref calldata);
    // ...
    let symbol_selector = selector!("symbol");
    let mut symbol = call_contract_syscall(token, symbol_selector, array![].span())
        .unwrap_syscall();
    calldata = deserialize_and_append(symbol, calldata);

    // @audit-issue The decimals() function is called directly, but it is optional in the ERC20 standard.
    dispatcher.decimals().serialize(ref calldata);
    calldata.span()
}
```

The problem arises because the ERC20 token standard specifies that the `decimals()` function is `OPTIONAL`. While many tokens implement it for usability, its presence is not guaranteed.

Consequently, any valid ERC20-compliant token that does not implement the `decimals()` function will be incompatible with the protocol. When the `deployment_message_payload(...)` function is called with such a token, the direct call to `dispatcher.decimals()` will fail, causing the transaction to revert. This restricts the variety of tokens the platform can support and may lead to unexpected failures for users.

Recommendation(s): Consider whether the `decimals()` value is strictly necessary for the `deployment_message_payload(...)` function's intended purpose. If the `decimals()` information is not critical for the core functionality, consider removing the call to `dispatcher.decimals()` to ensure broader compatibility with all ERC20 tokens, including those that do not implement the optional `decimals()` function.

Status: Acknowledged

Update from the client: Acknowledged. We decide on to support tokens that support `decimals()`.

6.5 [Info] Token activation can fail due to a race condition

File(s): src/bridge/token_bridge.cairo

Description: To enable deposits for a new token, it must first be deployed on the Appchain. This process is initiated on Starknet, and the token's status in the `token_bridge` contract is set to `Pending`. The `check_deployment_status(...)` function is then called within deposit-related functions to check if the deployment message has been successfully consumed on the Appchain and verified on Starknet. If so, the message is marked as `SEALED` and the token's status is updated to `Active`, enabling deposits.

The `check_deployment_status(...)` function includes a timeout mechanism. If a `Pending` token's deployment message is not `SEALED` within the `pending_deployment_expiration` period (currently 5 days), its status is reverted to `Unknown`. A race condition can occur if an Appchain's sequencer is inactive for more than 5 days and then comes back online to process the deployment message.

In this scenario, the token will be successfully deployed on the Appchain, but its status on the Starknet `token_bridge` will have already reverted to `Unknown`. Any subsequent attempts to re-register the token will fail on the Appchain with a `TOKEN_ALREADY_EXISTS` error. Since there is no mechanism to remove the token on the Appchain, this leads to a permanent state inconsistency, effectively blocking all future deposits for that token through the bridge. While this is unlikely for the main Starknet `L1 -> L2` bridge, it poses a risk for newer Appchains that may experience sequencer downtime.

```
fn check_deployment_status(ref self: ContractState, token: ContractAddress) {
    self.pausable.assert_not_paused();

    let settings = self.token_settings.read(token);

    if (settings.token_status != TokenStatus::Pending) {
        return;
    }

    let message_status = self
        .messaging_contract
        .read()
        .sn_to_appchain_messages(settings.deployment_message_hash);

    if (message_status == MessageToAppchainStatus::Sealed) {
        let new_settings = TokenSettings { token_status: TokenStatus::Active, ..settings };
        self.token_settings.write(token, new_settings);
        self.emit(TokenActivated { token });
    } else if (get_block_timestamp() > settings.pending_deployment_expiration) {
        let new_settings = TokenSettings { token_status: TokenStatus::Unknown, ..settings };
        // @audit-issue If the Appchain processes the deployment after this timeout, the token
        // will be deployed on the Appchain but remain `Unknown` on Starknet, blocking it permanently.
        self.token_settings.write(token, new_settings);
    }
}
```

Recommendation(s): Consider revisiting the token activation mechanism to handle this edge case. One approach could be to utilize the Piltover's message cancellation mechanism to ensure that the token wasn't activated at the smart contract level.

Status: Fixed

Update from the client: Fixed in 6025283.

Update from the Nethermind Security: The latest changes, contain issues that need to be addressed:

- The condition `message_status != MessageToAppchainStatus::Cancelling && get_block_timestamp() > settings.pending_deployment_expiration` is problematic. This logic allows the code to enter this branch even if the message has already been cancelled, preventing it from reaching the `MessageToAppchainStatus::Cancelled` handling. The `start_message_cancellation` function should only be invoked when the message status is `Pending`;
- The current implementation is missing a crucial step. After `start_message_cancellation` is called and the cancellation delay has passed, there should be a call to the `cancel_message` function. Without this, the message cancellation process remains incomplete;

Update from the client: Fixed in 091c873.

6.6 [Info] Token settings are not cleared upon unsuccessful token activation

File(s): src/bridge/token_bridge.cairo

Description: The token_bridge contract manages the enrollment of new tokens. When a user interacts with a token, the check_deployment_status(...) function is called to verify if the token has been successfully activated on the Appchain. Activation is confirmed if the corresponding deployment message from Starknet has been sealed.

If the deployment message is not sealed within the pending_deployment_expiration window (currently 5 days), the token activation is considered to have failed. In this scenario, the function correctly reverts the token_status to TokenStatus::Unknown. However, it fails to clear the deployment_message_hash and pending_deployment_expiration from the failed enrollment attempt.

While other settings like max_total_balance may be intended to persist as they can be set by the governor independently of the token status, the leftover message hash and expiration time represent stale data. This "dirty" state could be misleading for off-chain tooling that queries these values and could introduce subtle bugs in future contract iterations if new logic does not account for an Unknown token having leftover deployment data.

```
fn check_deployment_status(ref self: ContractState, token: ContractAddress) {
    // ...
    if (message_status == MessageToAppchainStatus::Sealed) {
        // ...
    } else if (get_block_timestamp() > settings.pending_deployment_expiration) {
        // @audit-issue Only the token_status is updated to Unknown. Other settings from the failed
        // deployment, like deployment_message_hash and pending_deployment_expiration,
        // persist, leaving stale data in storage.
        let new_settings = TokenSettings { token_status: TokenStatus::Unknown, ..settings };
        self.token_settings.write(token, new_settings);
    }
}
```

Recommendation(s): Consider clearing all relevant settings for a token when its deployment expires and it is marked as Unknown.

Status: Fixed

Update from the client: Fixed in e0672e1.

6.7 [Info] Tokens in the pending state cannot be blocked or deactivated

File(s): src/bridge/token_bridge.cairo

Description: The token_bridge contract provides several security roles to manage the lifecycle of tokens. The token_admin can move a token from an Active state to Deactivated or from an Unknown state to Blocked, preventing further deposits in both cases.

However, a gap exists in this security model for tokens that are in the Pending state - the period after enrollment but before successful activation. During this window, there is no direct mechanism for the token_admin to block the token. Instead, they must wait for the pending period to resolve. This requires actively monitoring the token's status until it either becomes Active (if deployment succeeds) or reverts to Unknown (if it fails), at which point it can finally be blocked or deactivated. This reactive approach is inefficient and introduces a risk that users might interact with a problematic token before it can be contained.

The only pre-emptive action available is for a higher-privileged app_governor to set the token's max_total_balance to 1 (lowest possible value). This creates an asymmetry in privileges, requiring escalation for a situation that a security-focused role like token_admin should be able to handle.

Recommendation(s): Consider introducing a mechanism to allow a privileged role, such as the token_admin, to block a token while it is in the Pending state. This could be achieved by adding a new token status to prevent the token from becoming active and to halt any associated actions.

Status: Acknowledged

Update from the client: Fixing this causes unnecssary complication in the design. set_max_total_balance seems fine to be used in case a situation arises.

6.8 [Best Practices] Incorrect error message in the `decrease_withdrawal_limit(...)` function

File(s): src/bridge/token_bridge.cairo

Description: The `decrease_withdrawal_limit(...)` function is an administrative function that can be called by the security agent to lower the daily withdrawal limit percentage for a given token.

The function correctly asserts that the new limit is strictly less than the current one. However, the error message provided in the `assert` statement is `Errors::NEW_LIMIT_MUST_BE_GREATER`, which is the opposite of the required condition.

This incorrect error message can be misleading for the caller, when attempting to adjust the withdrawal limits.

```
fn decrease_withdrawal_limit(
    ref self: ContractState, token: ContractAddress, daily_withdrawal_limit_pct: u8,
) {
    self.bridge_access_control.assert_only_security_agent();

    let current_pct = self.withdrawal.get_daily_withdrawal_limit_pct(token);

    // @audit-issue The error message is incorrect. The new limit must be smaller.
    assert(daily_withdrawal_limit_pct < current_pct, Errors::NEW_LIMIT_MUST_BE_GREATER);

    self.withdrawal.write_daily_withdrawal_limit_pct(token, daily_withdrawal_limit_pct);
    // ...
}
```

Recommendation(s): Consider changing the error message to accurately reflect that the new daily withdrawal limit percentage must be smaller than the current one.

Status: Fixed

Update from the client: Fixed in 2946788.

6.9 [Best Practices] Missing event emission during unsuccessful token activation

File(s): src/bridge/token_bridge.cairo

Description: The `check_deployment_status(...)` function emits a `TokenActivated` event when a token is successfully activated. However, when activation fails due to an expired deployment, the token status is reverted to `Unknown` without emitting a corresponding event. The absence of an event for this state change makes it difficult for off-chain services to track unsuccessful token activations.

```
fn check_deployment_status(ref self: ContractState, token: ContractAddress) {
    // ...
    if (message_status == MessageToAppchainStatus::Sealed) {
        // @audit Event is emitted on successful activation.
        self.emit(TokenActivated { token });
        // ...
    } else if (get_block_timestamp() > settings.pending_deployment_expiration) {
        // @audit-issue No event is emitted when token activation fails and its status is reverted.
        let new_settings = TokenSettings { token_status: TokenStatus::Unknown, ..settings };
        self.token_settings.write(token, new_settings);
    }
}
```

Recommendation(s): Consider emitting a dedicated event when a token's deployment expires and its status is reverted to `Unknown`.

Status: Fixed

Update from the client: Fixed in 6025283.

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 Token Bridge documentation

The `Token Bridge` team has provided a comprehensive walkthrough of the project in the kick-off call, and the README includes a detailed explanation of the intended functionalities. Moreover, the team addressed all questions and concerns raised by the Nethermind Security team, providing valuable insights and a comprehensive understanding of the project's technical aspects.

8 Test Suite Evaluation

8.1 Tests Output

```
> scarb test

Collected 101 test(s) from starknet_bridge package
Running 67 test(s) from tests/
[PASS] starknet_bridge_tests::deposit_reclaim_test::deposit_reclaim_paused (l1_gas: ~0, l1_data_gas: ~3840, l2_gas:
↪  ~6282880)
[PASS] starknet_bridge_tests::deposit_test::deposit_cancel_request_ok (l1_gas: ~0, l1_data_gas: ~3936, l2_gas: ~9346880)
[PASS] starknet_bridge_tests::deposit_reclaim_test::deposit_reclaim_not_cancelled (l1_gas: ~0, l1_data_gas: ~4032,
↪  l2_gas: ~8328640)
[PASS] starknet_bridge_tests::deposit_test::deposit_should_activate_token (l1_gas: ~0, l1_data_gas: ~3936, l2_gas:
↪  ~8648640)
[PASS] starknet_bridge_tests::deposit_test::deposit_with_message_empty_message_ok (l1_gas: ~0, l1_data_gas: ~3936,
↪  l2_gas: ~8919360)
[PASS] starknet_bridge_tests::deposit_reclaim_test::deposit_reclaim_ok (l1_gas: ~0, l1_data_gas: ~4032, l2_gas:
↪  ~11086080)
[PASS] starknet_bridge_tests::deposit_reclaim_test::deposit_reclaim_different_user (l1_gas: ~0, l1_data_gas: ~4128,
↪  l2_gas: ~9146880)
[PASS] starknet_bridge_tests::deposit_reclaim_test::deposit_reclaim_delay_not_reached (l1_gas: ~0, l1_data_gas: ~4128,
↪  l2_gas: ~9026880)
[PASS] starknet_bridge_tests::deposit_reclaim_test::deposit_reclaim_with_message_different_user (l1_gas: ~0,
↪  l1_data_gas: ~4128, l2_gas: ~9368320)
[PASS] starknet_bridge_tests::deposit_test::deposit_should_activate_token_pending (l1_gas: ~0, l1_data_gas: ~3840,
↪  l2_gas: ~4892160)
[PASS] starknet_bridge_tests::deposit_test::deposit_cancel_request_paused (l1_gas: ~0, l1_data_gas: ~3840, l2_gas:
↪  ~6282880)
[PASS] starknet_bridge_tests::deposit_test::deposit_insufficient_balance (l1_gas: ~0, l1_data_gas: ~3840, l2_gas:
↪  ~6188480)
[PASS] starknet_bridge_tests::deposit_test::deposit_deactivated (l1_gas: ~0, l1_data_gas: ~3840, l2_gas: ~6442880)
[PASS] starknet_bridge_tests::deposit_test::deposit_insufficient_allowance (l1_gas: ~0, l1_data_gas: ~3840, l2_gas:
↪  ~6027520)
[PASS] starknet_bridge_tests::deposit_test::deposit_paused (l1_gas: ~0, l1_data_gas: ~3840, l2_gas: ~6282880)
[PASS] starknet_bridge_tests::deposit_test::deposit_ok (l1_gas: ~0, l1_data_gas: ~3936, l2_gas: ~8808640)
[PASS] starknet_bridge_tests::pause_bridge_test::is_not_paused_on_deployment_ok (l1_gas: ~0, l1_data_gas: ~3744,
↪  l2_gas: ~5987520)
[PASS] starknet_bridge_tests::pause_bridge_test::pause_ok (l1_gas: ~0, l1_data_gas: ~3840, l2_gas: ~6522880)
[PASS] starknet_bridge_tests::pause_bridge_test::pause_bridge_only_security_agent (l1_gas: ~0, l1_data_gas: ~3744,
↪  l2_gas: ~6067520)
[PASS] starknet_bridge_tests::token_bridge_test::configure_permissioned_enrollment_not_app_governor (l1_gas: ~0,
↪  l1_data_gas: ~2496, l2_gas: ~2252480)
[PASS] starknet_bridge_tests::token_bridge_test::configure_permissioned_enrollment_ok (l1_gas: ~0, l1_data_gas: ~2592,
↪  l2_gas: ~2627840)
[PASS] starknet_bridge_tests::token_bridge_test::constructor_ok (l1_gas: ~0, l1_data_gas: ~2496, l2_gas: ~2292480)
[PASS] starknet_bridge_tests::pause_bridge_test::unpause_ok (l1_gas: ~0, l1_data_gas: ~3744, l2_gas: ~6978240)
[PASS] starknet_bridge_tests::pause_bridge_test::unpause_bridge_only_security_admin (l1_gas: ~0, l1_data_gas: ~3840,
↪  l2_gas: ~6442880)
[PASS] starknet_bridge_tests::token_bridge_test::is_enrollment_permissionless_ok (l1_gas: ~0, l1_data_gas: ~2496,
↪  l2_gas: ~2292480)
[PASS] starknet_bridge_tests::token_bridge_test::set_appchain_bridge_not_app_governor (l1_gas: ~0, l1_data_gas: ~2496,
↪  l2_gas: ~2372480)
[PASS] starknet_bridge_tests::token_bridge_test::set_appchain_bridge_paused (l1_gas: ~0, l1_data_gas: ~2592, l2_gas:
↪  ~3043200)
[PASS] starknet_bridge_tests::token_bridge_test::set_appchain_bridge_ok (l1_gas: ~0, l1_data_gas: ~2496, l2_gas:
↪  ~2707840)
[PASS] starknet_bridge_tests::token_bridge_test::set_max_total_balance_not_app_governor (l1_gas: ~0, l1_data_gas:
↪  ~2496, l2_gas: ~2292480)
[PASS] starknet_bridge_tests::token_bridge_test::set_max_total_balance_ok (l1_gas: ~0, l1_data_gas: ~2592, l2_gas:
↪  ~2683200)
[PASS] starknet_bridge_tests::token_bridge_test::set_max_total_balance_paused (l1_gas: ~0, l1_data_gas: ~2688, l2_gas:
↪  ~2978560)
[PASS] starknet_bridge_tests::withdraw_test::withdraw_incorrect_recipient (l1_gas: ~0, l1_data_gas: ~4224, l2_gas:
↪  ~9009600)
[PASS] starknet_bridge_tests::withdraw_test::withdraw_limit_reached (l1_gas: ~0, l1_data_gas: ~4224, l2_gas: ~9460800)
[PASS] starknet_bridge_tests::withdrawal_limit_bridge_test::decrease_withdrawal_limit_not_security_agent (l1_gas: ~0,
↪  l1_data_gas: ~2496, l2_gas: ~2252480)
[PASS] starknet_bridge_tests::withdraw_test::withdraw_ok (l1_gas: ~0, l1_data_gas: ~3936, l2_gas: ~10403200)
```

```
[PASS] starknet_bridge_tests::withdrawal_limit_bridge_test::decrease_withdrawal_limit_ok (l1_gas: ~0, l1_data_gas:
↪  ~2592, l2_gas: ~2863680)
[PASS] starknet_bridge_tests::withdrawal_limit_bridge_test::disable_withdrawal_limit_limit_not_applied (l1_gas: ~0,
↪  l1_data_gas: ~2496, l2_gas: ~2332480)
[PASS] starknet_bridge_tests::withdrawal_limit_bridge_test::decrease_withdrawal_limit_paused (l1_gas: ~0, l1_data_gas:
↪  ~2688, l2_gas: ~3199040)
[PASS] starknet_bridge_tests::withdraw_test::withdraw_paused (l1_gas: ~0, l1_data_gas: ~4128, l2_gas: ~9104960)
[PASS] starknet_bridge_tests::withdrawal_limit_bridge_test::disable_withdrawal_limit_not_security_admin (l1_gas: ~0,
↪  l1_data_gas: ~2592, l2_gas: ~2743680)
[PASS] starknet_bridge_tests::withdrawal_limit_bridge_test::increase_withdrawal_limit_not_security_admin (l1_gas: ~0,
↪  l1_data_gas: ~2496, l2_gas: ~2252480)
[PASS] starknet_bridge_tests::withdrawal_limit_bridge_test::disable_withdrawal_limit_ok (l1_gas: ~0, l1_data_gas:
↪  ~2496, l2_gas: ~3514880)
[PASS] starknet_bridge_tests::withdrawal_limit_bridge_test::increase_withdrawal_limit_ok (l1_gas: ~0, l1_data_gas:
↪  ~2592, l2_gas: ~3274880)
[PASS] starknet_bridge_tests::withdrawal_limit_bridge_test::disable_withdrawal_limit_paused (l1_gas: ~0, l1_data_gas:
↪  ~2592, l2_gas: ~3850240)
[PASS] starknet_bridge_tests::withdrawal_limit_bridge_test::increase_withdrawal_limit_paused (l1_gas: ~0, l1_data_gas:
↪  ~2688, l2_gas: ~3610240)
[PASS] starknet_bridge_tests::withdrawal_limit_bridge_test::is_withdrawal_limit_applied_ok (l1_gas: ~0, l1_data_gas:
↪  ~2592, l2_gas: ~2743680)
[PASS] starknet_bridge_tests::deposit_test::deposit_with_message_ok (l1_gas: ~0, l1_data_gas: ~3936, l2_gas: ~8919360)
[PASS] starknet_bridge_tests::deposit_test::deposit_with_message_insufficient_balance (l1_gas: ~0, l1_data_gas: ~3840,
↪  l2_gas: ~6188480)
[PASS] starknet_bridge_tests::deposit_test::deposit_with_message_insufficient_allowance (l1_gas: ~0, l1_data_gas:
↪  ~3840, l2_gas: ~6027520)
[PASS] starknet_bridge_tests::deposit_test::deposit_with_message_cancel_request_ok (l1_gas: ~0, l1_data_gas: ~3936,
↪  l2_gas: ~9608320)
[PASS] starknet_bridge_tests::deposit_test::deposit_with_message_paused (l1_gas: ~0, l1_data_gas: ~3840, l2_gas:
↪  ~6282880)
[PASS] starknet_bridge_tests::enroll_token_test::enroll_token_paused (l1_gas: ~0, l1_data_gas: ~3360, l2_gas: ~3349760)
[PASS] starknet_bridge_tests::deposit_test::deposit_with_message_cancel_request_no_deposit (l1_gas: ~0, l1_data_gas:
↪  ~3840, l2_gas: ~5987520)
[PASS] starknet_bridge_tests::enroll_token_test::enroll_token_already_enrolled (l1_gas: ~0, l1_data_gas: ~3840, l2_gas:
↪  ~4732160)
[PASS] starknet_bridge_tests::enroll_token_test::enroll_token_permissioned_not_token_admin (l1_gas: ~0, l1_data_gas:
↪  ~3456, l2_gas: ~3309760)
[PASS] starknet_bridge_tests::enroll_token_test::enroll_token_ok (l1_gas: ~0, l1_data_gas: ~3744, l2_gas: ~5412160)
[PASS] starknet_bridge_tests::deposit_test::deposit_with_message_cancel_request_paused (l1_gas: ~0, l1_data_gas: ~3840,
↪  l2_gas: ~6282880)
[PASS] starknet_bridge_tests::deposit_test::deposit_with_message_deactivated (l1_gas: ~0, l1_data_gas: ~3840, l2_gas:
↪  ~6442880)
[PASS] starknet_bridge_tests::enroll_token_test::enroll_token_permissioned_ok (l1_gas: ~0, l1_data_gas: ~3840, l2_gas:
↪  ~5827520)
[PASS] starknet_bridge_tests::deposit_reclaim_test::deposit_with_message_reclaim_paused (l1_gas: ~0, l1_data_gas:
↪  ~3840, l2_gas: ~6282880)
[PASS] starknet_bridge_tests::deposit_reclaim_test::deposit_with_message_reclaim_ok (l1_gas: ~0, l1_data_gas: ~4032,
↪  l2_gas: ~11418240)
[PASS] starknet_bridge_tests::deposit_reclaim_test::deposit_wtih_message_reclaim_not_cancelled (l1_gas: ~0,
↪  l1_data_gas: ~4032, l2_gas: ~8439360)
[PASS] starknet_bridge_tests::enroll_token_test::enroll_token_nonce_not_updated (l1_gas: ~0, l1_data_gas: ~2496,
↪  l2_gas: ~2269120)
[PASS] starknet_bridge_tests::deposit_test::deposit_cancel_request_different_user (l1_gas: ~0, l1_data_gas: ~4032,
↪  l2_gas: ~8328640)
[PASS] starknet_bridge_tests::deposit_test::deposit_cancel_request_no_deposit (l1_gas: ~0, l1_data_gas: ~3840, l2_gas:
↪  ~5987520)
[PASS] starknet_bridge_tests::deposit_reclaim_test::deposit_with_message_reclaim_delay_not_reached (l1_gas: ~0,
↪  l1_data_gas: ~4128, l2_gas: ~9248320)
[PASS] starknet_bridge_tests::deposit_test::deposit_with_message_cancel_request_different_user (l1_gas: ~0,
↪  l1_data_gas: ~4032, l2_gas: ~8439360)
Running 34 test(s) from src/
[PASS] starknet_bridge::bridge::tests::messaging_test::consume_message_bridge_unset (l1_gas: ~0, l1_data_gas: ~0,
↪  l2_gas: ~40000)
[PASS] starknet_bridge::access_control::tests::access_control_test::test_governance_admin_cannot_renounce (l1_gas: ~0,
↪  l1_data_gas: ~1344, l2_gas: ~1267200)
[PASS] starknet_bridge::bridge::tests::messaging_test::consume_message_zero_recipient (l1_gas: ~0, l1_data_gas: ~160,
↪  l2_gas: ~40000)
[PASS] starknet_bridge::bridge::tests::messaging_test::consume_message_no_message (l1_gas: ~0, l1_data_gas: ~1696,
↪  l2_gas: ~1227200)
```

```
[PASS] starknet_bridge::bridge::tests::messaging_test::consume_message_ok (l1_gas: ~0, l1_data_gas: ~1696, l2_gas:
↪  ~1618880)
calldata [3229811236586461276790806733073987758974063646349769890211994918419655038703, 0, 1431520323, 4, 0,
↪  1431520323, 4, 18]
[PASS] starknet_bridge::access_control::tests::access_control_test::test_access_control_roles (l1_gas: ~0, l1_data_gas:
↪  ~1344, l2_gas: ~3267200)
[PASS] starknet_bridge::bridge::tests::messaging_test::deposit_message_payload_with_message_false_ok (l1_gas: ~0,
↪  l1_data_gas: ~0, l2_gas: ~40000)
[PASS] starknet_bridge::bridge::tests::messaging_test::deploy_message_payload_ok (l1_gas: ~0, l1_data_gas: ~768,
↪  l2_gas: ~4001920)
[PASS] starknet_bridge::bridge::tests::token_actions_test::is_servicing_token_ok (l1_gas: ~0, l1_data_gas: ~160,
↪  l2_gas: ~280000)
[PASS] starknet_bridge::bridge::tests::messaging_test::send_deploy_message_bridge_unset (l1_gas: ~0, l1_data_gas: ~0,
↪  l2_gas: ~40000)
[PASS] starknet_bridge::bridge::tests::token_actions_test::reactivate_token_not_owner (l1_gas: ~0, l1_data_gas: ~160,
↪  l2_gas: ~240000)
[PASS] starknet_bridge::bridge::tests::token_actions_test::reactivate_token_ok (l1_gas: ~0, l1_data_gas: ~256, l2_gas:
↪  ~495360)
[PASS] starknet_bridge::bridge::tests::messaging_test::send_deposit_message_bridge_unset (l1_gas: ~0, l1_data_gas: ~0,
↪  l2_gas: ~40000)
[PASS] starknet_bridge::bridge::tests::token_actions_test::reactivate_token_not_deactivated (l1_gas: ~0, l1_data_gas:
↪  ~256, l2_gas: ~320000)
[PASS] starknet_bridge::bridge::tests::token_actions_test::unblock_token_ok (l1_gas: ~0, l1_data_gas: ~160, l2_gas:
↪  ~495360)
[PASS] starknet_bridge::bridge::tests::token_actions_test::unblock_token_not_owner (l1_gas: ~0, l1_data_gas: ~256,
↪  l2_gas: ~240000)
[PASS] starknet_bridge::timelock::tests::timelock_test::test_timelock_deploy (l1_gas: ~0, l1_data_gas: ~768, l2_gas:
↪  ~985280)
[PASS] starknet_bridge::bridge::tests::token_actions_test::unblock_token_not_blocked (l1_gas: ~0, l1_data_gas: ~256,
↪  l2_gas: ~320000)
[PASS] starknet_bridge::withdrawal_limit::tests::withdrawal_limit_test::consume_withdrawal_quota_limit_exceeded
↪  (l1_gas: ~0, l1_data_gas: ~1056, l2_gas: ~2179200)
[PASS] starknet_bridge::withdrawal_limit::tests::withdrawal_limit_test::consume_withdrawal_quota_ok (l1_gas: ~0,
↪  l1_data_gas: ~1152, l2_gas: ~2929920)
[PASS] starknet_bridge::bridge::tests::token_actions_test::block_token_ok (l1_gas: ~0, l1_data_gas: ~256, l2_gas:
↪  ~375360)
[PASS] starknet_bridge::withdrawal_limit::tests::withdrawal_limit_test
::get_remaining_withdrawal_quota_should_reset_after_1_day_ok (l1_gas: ~0, l1_data_gas: ~1152, l2_gas: ~3129920)
[PASS] starknet_bridge::withdrawal_limit::tests::withdrawal_limit_test::write_daily_withdrawal_limit_pct_ok (l1_gas:
↪  ~0, l1_data_gas: ~192, l2_gas: ~495360)
[PASS] starknet_bridge::bridge::tests::token_actions_test::get_status_ok (l1_gas: ~0, l1_data_gas: ~160, l2_gas:
↪  ~440000)
[PASS] starknet_bridge::withdrawal_limit::tests::withdrawal_limit_test::get_remaining_withdrawal_quota_ok (l1_gas: ~0,
↪  l1_data_gas: ~1056, l2_gas: ~2219200)
[PASS] starknet_bridge::bridge::tests::token_actions_test::deactivate_token_not_active (l1_gas: ~0, l1_data_gas: ~160,
↪  l2_gas: ~200000)
[PASS] starknet_bridge::withdrawal_limit::tests::withdrawal_limit_test::write_daily_withdrawal_limit_pct_too_high
↪  (l1_gas: ~0, l1_data_gas: ~192, l2_gas: ~535360)
[PASS] starknet_bridge::bridge::tests::token_actions_test::enroll_token_blocked (l1_gas: ~0, l1_data_gas: ~352, l2_gas:
↪  ~360000)
[PASS] starknet_bridge::bridge::tests::token_actions_test::deactivate_token_not_owner (l1_gas: ~0, l1_data_gas: ~160,
↪  l2_gas: ~120000)
[PASS] starknet_bridge::bridge::tests::token_actions_test::block_token_not_unknown (l1_gas: ~0, l1_data_gas: ~256,
↪  l2_gas: ~320000)
[PASS] starknet_bridge::bridge::tests::token_actions_test::block_token_not_owner (l1_gas: ~0, l1_data_gas: ~160,
↪  l2_gas: ~120000)
[PASS] starknet_bridge::bridge::tests::token_actions_test::deactivate_token_ok (l1_gas: ~0, l1_data_gas: ~1600, l2_gas:
↪  ~1402560)
[PASS] starknet_bridge::bridge::tests::messaging_test::send_deploy_message_ok (l1_gas: ~0, l1_data_gas: ~2656, l2_gas:
↪  ~3406400)
[PASS] starknet_bridge::bridge::tests::messaging_test::send_deposit_message_ok (l1_gas: ~0, l1_data_gas: ~1888, l2_gas:
↪  ~1829120)
Tests: 101 passed, 0 failed, 0 skipped, 0 ignored, 0 filtered out
```

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.