Audit Report

# Router Asset Bridge

v1.2

April 30, 2024

# Table of Contents

## License

## Disclaimer

This audit has been performed by

**Oak Security GmbH**

https://oaksecurity.io/
info@oaksecurity.io

# Introduction

## Purpose of This Report

Oak Security has been engaged by Kailaasa Infotech Pte Ltd to perform a security audit of the Router Asset Bridge.

The objectives of the audit are as follows:

1. Determine the correct functioning of the protocol, in accordance with the project specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine smart contract bugs, which might lead to unexpected behavior.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

## Codebase Submitted for the Audit

The audit has been performed on the following target:

| | |
|---|---|
| Repository | https://github.com/router-protocol/asset-bridge-contracts |
| Commit | `4fc1add571ea0525e8f1fac28ef8ced8d03894ca` |
| Scope | All contracts were in the scope, excluding the test files and contracts inside the `near/contracts/sequencer-staking` directory. |
| Fixes verified at commit | `6a2f6b3d719016960b56f5b68b393b06a10bf967`<br><br>Note that only fixes to the issues described in this report have been reviewed at this commit. Any further changes such as additional features have not been reviewed. |

## Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line-by-line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under-/overflow issues
    c. Key management vulnerabilities
4. Report preparation

## Functionality Overview

Router Chain is a layer one blockchain focusing on blockchain interoperability, enabling cross-chain communication with CosmWasm middleware contracts.

# How to Read This Report

This report classifies the issues found into the following severity categories:

| Severity | Description |
|---|---|
| Critical | A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service. |
| Major | A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service. |
| Minor | A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies. |
| Informational | Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share. |

The status of an issue can be one of the following: **Pending, Acknowledged**, or **Resolved**.

Note that audits are an important step to improving the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than in a security audit and vice versa.

## Code Quality Criteria

The auditor team assesses the codebase's code quality criteria as follows:

| Criteria | Status | Comment |
|---|---|---|
| Code complexity | **Medium** | - |
| Code readability and clarity | **Low-Medium** | There are many outstanding `TODO` comments throughout the codebase, along with unimplemented functionalities, such as excess fee refunds from IBC acknowledgments and incomplete role mechanisms. |
| Level of documentation | **Medium-High** | The client provided insightful diagrams and detailed documentation. |
| Test coverage | **Low** | Most of the test cases fail and lack full coverage. |

## Summary of Findings

| No | Description | Severity | Status |
|---|---|---|---|
| 1 | Attackers can front-run transactions to steal user funds after approving allowances | Critical | Resolved |
| 2 | Attackers can profit by depositing different tokens and withdrawing as native tokens | Critical | Resolved |
| 3 | Attackers can register and deregister their accounts for profit | Critical | Resolved |
| 4 | Inability to decode request packet payload due to incorrect indexes results in failed cross-chain requests | Critical | Resolved |
| 5 | Refunding fails due to `set_execute_revert_record` function | Critical | Resolved |
| 6 | NEAR tokens are incorrectly transferred to the `gateway` contract instead of being escrowed in the `AssetBridge` contract | Critical | Resolved |
| 7 | Fungible token refunds will fail, causing a loss of funds for users | Critical | Resolved |
| 8 | Payable keyword is not specified on functions that expect funds to be sent | Major | Resolved |
| 9 | Native funds cannot be attached on `ft_on_transfer` calls | Major | Resolved |
| 10 | Unstaking liquidity token fails due to incorrect token parameters | Major | Resolved |
| 11 | Failed cross-chain requests cannot be retried | Major | Resolved |
| 12 | Unsafe conversion from `uint256` to `uint128` in `asset-bridge` middleware | Major | Resolved |
| 13 | The same gas limit is used for all destination chains, resulting in out-of-gas errors | Major | Resolved |
| 14 | Unable to process refund requests on the source chain due to missing parameters | Major | Resolved |
| 15 | The reentrancy lock mechanism in the NEAR `AssetBridge` contract can be abused to grief the contract | Major | Resolved |

| | | | |
|---|---|---|---|
| 16 | Native token transfers always fail due to improper balance check query | Major | Resolved |
| 17 | Lowercasing case-sensitive addresses causes unexpected behavior and incorrect outbound calls | Major | Resolved |
| 18 | Unintended refund due to `TEMP_REVERT_STATE` not being cleared and incorrectly calling of `revert_inbound_to_src_chain` on minting gas tokens | Major | Resolved |
| 19 | Callback functions may run out of gas, resulting in inconsistent states of the NEAR `asset-bridge` contract | Major | Resolved |
| 20 | Incorrectly determining the available NEAR balance of the `AssetBridge` contract causes transactions to fail | Major | Resolved |
| 21 | The amount of minted gas tokens can exceed the actual charged fee amount | Major | Resolved |
| 22 | Panic in handler callback causes denial of service | Major | Resolved |
| 23 | The `asset-bridge` middleware contract owner can mint unlimited tokens that were previously burned | Minor | Resolved |
| 24 | Users lose accidentally sent native tokens when staking ERC-20 tokens | Minor | Resolved |
| 25 | Inaccurate `BURNED_GAS_TOKENS` accounting in case of errors | Minor | Resolved |
| 26 | `AssetBridge` Solidity contract is unable to transfer certain ERC-20 tokens | Minor | Resolved |
| 27 | `AssetBridge` Solidity contact is not compatible with ERC-20 tokens that charge a transfer fee | Minor | Acknowledged |
| 28 | Incomplete pause functionality in the codebase | Minor | Resolved |
| 29 | Potential precision loss for values larger than `2^53-1` | Minor | Resolved |
| 30 | NEAR contracts' initialization process can be front-run | Minor | Resolved |
| 31 | The recipient will receive the contract's NEAR balance instead of the specified amount | Minor | Resolved |
| 32 | Incorrect token price decimal scaling may result in incorrect fee calculation | Minor | Resolved |

| | | | |
|---|---|---|---|
| 33 | Zero address cannot be used to convert native funds | Minor | Resolved |
| 34 | Incorrect token refunds to the recipient instead of the sender | Minor | Resolved |
| 35 | Incorrect sender assignment in NEAR `asset-bridge` contract | Minor | Resolved |
| 36 | Lack of entry point to remove outdated storage entries | Minor | Partially Resolved |
| 37 | Users might unintentionally lose fees if the gateway contract is not updated | Minor | Resolved |
| 38 | Potential out-of-gas error due to unbounded query iterations | Minor | Resolved |
| 39 | Missing liquidity token configuration results in a failed refund request and the inability to retry the request | Minor | Resolved |
| 40 | Usage of similar storage prefixes might cause collision | Minor | Resolved |
| 41 | Failed cross-chain request fees are borne by the `middleware` contract | Informational | Acknowledged |
| 42 | Potential version mismatch due to hardcoded value | Informational | Acknowledged |
| 43 | Inconsistency in nonce configuration for `AssetBridge` contract | Informational | Resolved |
| 44 | Incorrect vector length validation in `set_liquidity_pool` | Informational | Resolved |
| 45 | `nonReentrant` modifiers are not added for external calls | Informational | Resolved |
| 46 | Inconsistency in `srcChainId` validation | Informational | Resolved |
| 47 | Redundant `is_native` parameter | Informational | Resolved |
| 48 | Unneeded payable annotation | Informational | Resolved |
| 49 | Incorrect events emitted | Informational | Resolved |
| 50 | Incomplete resource setter role mechanism | Informational | Resolved |
| 51 | Token decimals are stored in both the asset bridge middleware and fee manager contracts, leading to potential inconsistencies | Informational | Acknowledged |

| | | | |
|---|---|---|---|
| 52 | Incorrect errors in NEAR `asset-bridge` contract | **Informational** | **Resolved** |
| 53 | Contracts should implement a two-step ownership transfer | **Informational** | **Acknowledged** |

# Detailed Findings

## 1. Attackers can front-run transactions to steal user funds after approving allowances

**Severity: Critical**

In `evm/contracts/AssetBridge.sol:602`, the `stake` function allows the owner to specify the `from` parameter, representing which user's allowance will be consumed, as seen in line `614`. This is problematic because users typically approve the contract to use their allowance before calling the contract.

An attacker can exploit this issue by front-running the victim after they approve the transaction but before they call the `AssetBridge` contract. The attacker will call the `stake` function by specifying the `from` parameter as the victim's address, the `to` parameter as the attacker's address, and the approved token address and amount. This will consume the victim's allowance and consume their funds, but the liquidity token is minted to the attacker.

This issue is also present in the `unstake` function in lines `625` and `629`. If the approved token is the liquidity token, the attacker can front-run and call the `unstake` function to consume the victim's liquidity token allowance and receive the underlying tokens in return, causing a loss of funds scenario.

### Recommendation

We recommend removing the `from` parameter and using `msg.sender` instead.

**Status: Resolved**

## 2. Attackers can profit by depositing different tokens and withdrawing as native tokens

**Severity: Critical**

In `evm/contracts/AssetBridge.sol:624-627`, the `unstake` function allows the caller to withdraw native tokens by specifying the `isSourceNative` parameter as `true`. This is problematic because attackers can provide non-ETH tokens and withdraw them as native tokens and profit from price differences.

For example, assume the attacker staked 10 TRX and received 10 TRX liquidity tokens. The price of TRX is $0.094, so the attacker's cost would be $0.94. The TRX token uses decimal values of 6.

The attacker then calls the `unstake` function with all 10 TRX and specifies the `isSourceNative` parameter as `true`. In decimal terms, the attacker sends 10*10^6 TRX and receives 10*10^6 wei in return, which translates to $197. After deducting the attacker's cost, the attacker gains a profit of $196.06 per transaction. This profit can be leveraged to a higher value by using tokens with lower prices and higher decimal values and repeatedly performing the attack until all wrapped native tokens are stolen in the contract.

Consequently, all wrapped native tokens in the contract can be stolen, causing a loss of funds scenario.

### Recommendation

We recommend removing the `isSourceNative` parameter and determining whether the `tokenAddress` is native by checking whether `_lpToContract[tokenAddress]` equals the WETH address.

**Status: Resolved**

## 3. Attackers can register and deregister their accounts for profit

**Severity: Critical**

In `near/contracts/token/src/lib.rs:109`, the `register` function registers the account in the fungible token contract after the caller paid 1 `yoctoNEAR`. This is problematic because [unregistering the account storage refunds the caller with `storage_balance_bounds().min.0`], which is 1250000000000000000000 `yoctoNEAR`.

Consequently, an attacker can steal the funds by repeatedly calling `register` and [`storage_unregister`] functions to profit from the account storage refunds, causing a loss of funds for the account owner.

This issue also exists in `near/contracts/lp-token/src/lib.rs:193`, with the difference that the user does not need to pay any funds to register their account.

### Recommendation

We recommend removing the function and letting users register accounts using the [`storage_deposit` function] instead.

**Status: Resolved**

## 4. Inability to decode request packet payload due to incorrect indexes results in failed cross-chain requests

**Severity: Critical**

In `near/contracts/asset-bridge/src/utils.rs`, the `decode_ireceive_deposit`, `decode_ireceive_execute_with_message`, and `decode_stake_payload` utility functions decode the received cross-chain request packet payload and return the decoded values.

However, in the following instances, incorrect indexes are used to access the decoded values, resulting in a panic since the expected and actual types do not match:

- Line `250` should parse `decoded_data[2]` for the tuple.
- Line `476` should parse `decoded_data[3]` for the instruction bytes.
- Line `618` should parse `decoded_data[3]` for the recipient.

Additionally, line `215` decodes the tuple as `destTokenAmount`, `destTokenAddress`, `isDestNative`, `depositNonce`, and `recipient`. However, in lines `258-305`, the tuple is accessed as `sender`, `src_token`, `src_amount`, and `deposit_nonce`. This is incorrect because the tuple's length and variable types are incorrectly used, causing the decoding to fail.

### Recommendation

We recommend using the correct indexes to access the decoded values.

**Status: Resolved**

## 5. Refunding fails due to `set_execute_revert_record` function

**Severity: Critical**

In `near/contracts/asset-bridge/src/setters.rs:34-40`, the `set_execute_revert_record` function tries to unwrap the value in the `execution_revert_completed` map and validates that it equals the provided status parameter in line `32`. This is problematic because a panic will occur for new key entries, and the assertion will fail if the status is updated to a different value for existing entries.

Firstly, the `i_receive` function in `near/contracts/asset-bridge/src/execution.rs:312` will fail because the destination chain ID and deposit nonce key entry are never stored before in the `execution_revert_completed` map. This will cause the `insert` function to return `None` in `near/contracts/asset-bridge/src/setters.rs:37` and panic when the unwrap function is called. Consequently, the refund mechanism will fail to work properly.

Secondly, the `unlock_ireceive_fn` function in `near/contracts/asset-bridge/src/execution.rs:146` calls `set_execute_revert_record` to revert the execute record to `false` in case promises fail so the action can be retried later. As the key entry's value was previously set to `true`, the `insert` function will return `Some(true)` and unwrap it. Consequently, the assertion will fail in `near/contracts/asset-bridge/src/setters.rs:39` because it expects the status to be `false`, which is incorrect.

#### Recommendation

We recommend removing the assertion in `near/contracts/asset-bridge/src/setters.rs:34`.

**Status: Resolved**

### 6. NEAR tokens are incorrectly transferred to the `gateway` contract instead of being escrowed in the `AssetBridge` contract

**Severity: Critical**

In `near/contracts/asset-bridge/src/call_gateway.rs:71`, the `i_send` function sends all the attached NEAR tokens to the gateway contract. This is incorrect because only `i_send_default_fee` should be sent to the gateway contract to cover the fees, while the remaining funds should remain in the contract.

Consequently, the `asset-bridge` contract will have insufficient liquidity for user token redemption, forcing users to wait for the contract owner to withdraw the funds from the gateway contract in order to unstake their LP tokens.

#### Recommendation

We recommend only attaching the required fee amount to the `i_send` call and escrowing the remaining NEAR tokens in the `AssetBridge` contract.

**Status: Resolved**

### 7. Fungible token refunds will fail, causing a loss of funds for users

**Severity: Critical**

In `near/contracts/asset-bridge/src/call_gateway.rs:164`, the `i_send_callback` function calls `ft_transfer` to refund tokens to the sender if the gateway contract's `i_send` call fails. This is problematic because no deposits are attached, causing the transaction to fail because the NEAR SDK requires one yoctoNEAR to be sent.

Consequently, the fungible token refund fails, causing a loss of funds for the sender.

#### Recommendation

We recommend adding `with_attached_deposit(1)` to line `164` when refunding tokens to users.

**Status: Resolved**

## 8. Payable keyword is not specified on functions that expect funds to be sent

**Severity: Major**

In `evm/contracts/AssetBridge.sol:194-196`, the `setDappMetadata` function allows the resource owner to call the `setDappMetadata` function in the `gateway` contract. Since no native funds are included, the transaction will fail [if the `iSendDefaultFee` fee amount in the gateway contract is greater than zero].

Besides that, this issue also exists in the NEAR `asset-bridge` contract in `near/contracts/asset-bridge/src/call_gateway.rs:21`. Since the payable keyword is not implemented, the `set_dapp_metadata` function cannot receive funds, causing the transaction to fail [if the `i_send_default_fee` fee amount is greater than zero].

Lastly, the `unstake` function in `near/contracts/asset-bridge/src/execution.rs:703` expects the caller to send three `yoctoNEAR`. This condition cannot be satisfied because the payable keyword is not implemented, causing the `unstake` function always to fail.

### Recommendation

We recommend implementing the payable keyword for the lines of code mentioned above. Additionally, the `setDappMetadata` function in `evm/contracts/AssetBridge.sol:194-196` should forward the funds to the `gateway` contract with `{ value: msg.value }`.

**Status: Resolved**

## 9. Native funds cannot be attached on `ft_on_transfer` calls

**Severity: Major**

In `near/contracts/asset-bridge/src/execution.rs:26-28`, the `ft_on_transfer` function implements the `payable` annotation because it expects native funds to be attached. This is problematic because when a user calls `ft_transfer_call` from a fungible token contract, [native funds cannot be attached], preventing the `asset-bridge` contract from working properly.

Firstly, the `i_send` function includes all attached deposits in `near/contracts/asset-bridge/src/call_gateway.rs:71` in order to [pay the `i_send_default_fee` fees in the gateway contract]. Since native funds cannot be attached, the actual fees sent will be zero, causing token transfers to fail if the fee amount is larger than zero.

Secondly, the `stake_token` function in `near/contracts/asset-bridge/src/execution.rs:669` expects the user to send three `yoctoNEAR`. As fungible tokens cannot include any deposits, the transaction will fail in line `109`, preventing users from staking the fungible token for liquidity tokens.

### Recommendation

We recommend introducing a deposit mechanism so users can deposit NEAR in advance before calling functions that require native funds to be attached. This can be achieved using an `UnorderedMap` that records how much funds each user deposited.

If fees need to be charged, the contract can deduct the user deposited amount and errors if it is insufficient, with the excess being withdrawable. For example, the `i_send` function should deduct the `i_send_default_fee` amount, while the `stake_token` function should deduct three `yoctoNEAR`.

**Status: Resolved**

## 10. Unstaking liquidity token fails due to incorrect token parameters

**Severity: Major**

In `evm/contracts/AssetBridge.sol:625`, `629`, and `630`, the `unstake` function uses incorrect tokens to call the `burnFrom` and `safeTransferFrom` functions. This function is called when users want to burn their liquidity token to redeem the underlying token.

Firstly, in lines `625` and `629`, the `_contractToLP` mapping is called on the `tokenAddress` parameter. Since the mapping is used to retrieve the underlying token's liquidity token, the transaction will fail because the provided `tokenAddress` is already the liquidity token. Hence, it is unnecessary to use the mapping.

Secondly, in line `630`, the `safeTransferFrom` function transfers the `tokenAddress` to the user after burning the liquidity tokens. This is incorrect because the liquidity tokens are sent instead of the underlying token, which can be retrieved via the `_lpToContract` mapping. Consequently, the transaction will fail because the contract holds no liquidity tokens after burning them.

### Recommendation

We recommend using `tokenAddress` in lines `625` and `629` and `_lpToContract[tokenAddress]` in line `630`.

**Status: Resolved**

## 11. Failed cross-chain requests cannot be retried

**Severity: Major**

The `execute_failed_request` function in `middleware/contracts/asset-bridge/src/execution.rs:493-510` handles the `ExecuteMsg::ExecuteFailedReq` message, which allows the owner of the CosmWasm `asset-bridge` middleware contract to retry and execute a previously failed cross-chain request.

To ensure that only failed requests can be retried, the status of the received acknowledgment is retrieved from the `ACK_STATUS` storage map. If the status is `false`, signaling a failed cross-chain request, the request is supposed to be retried.

However, this check is incorrectly performed in line `501` as the `assert_ne!` macro asserts that the status is **not** equal to `false`, which is the opposite of the intended behavior. As a result, failed cross-chain requests can not be retried and remain stuck.

### Recommendation

We recommend replacing the `assert_ne!` macro with the `assert_eq!` macro.

**Status: Resolved**

## 12. Unsafe conversion from `uint256` to `uint128` in `asset-bridge` middleware

**Severity: Major**

In `middleware/contracts/asset-bridge/src/handle_inbound.rs:129`, the `handle_evm_inbound_request` function decodes the `widget_id` from the received payload and attempts to convert the `uint` value to `uint128`. However, the Solidity `AssetBridge` contract defines the widget ID parameter as `uint256`. This results in a panic if the provided value is larger than `uint128` and halts the execution of the inbound request, causing the locked funds on the source chain to be unrecoverable by the user.

As this widget ID is provided by the user or by a widget integration, the ID may be incorrectly or maliciously set to a large value, causing the request to fail in the middleware contract.

### Recommendation

We recommend gracefully handling the conversion from `uint256` to `uint128` in all instances and returning an error if the conversion fails.

**Status: Resolved**

## 13. The same gas limit is used for all destination chains, resulting in out-of-gas errors

**Severity: Major**

In the following instances, the `asset-bridge` middleware contract determines the required gas limit for a cross-chain request by calling the `fetch_st_gas_limit` function:

- `middleware/contracts/asset-bridge/src/execution.rs:478`
- `middleware/contracts/asset-bridge/src/handle_inbound.rs:139`
- `middleware/contracts/asset-bridge/src/handle_inbound.rs:305`

However, the gas limit is not configured on a per-chain basis. Instead, the same gas limit is used for all chains. Consequently, cross-chain requests may fail on the destination chain due to an out-of-gas error.

Similarly, the gas limit in `middleware/contracts/asset-bridge/src/handle_revert.rs:64` is hardcoded to `150000`.

### Recommendation

We recommend specifying a chain-specific gas limit for each destination chain and refrain from hardcoding the gas limit.

**Status: Resolved**

## 14. Unable to process refund requests on the source chain due to missing parameters

**Severity: Major**

If processing the inbound request in the `asset-bridge` middleware results in an error, the `revert_inbound_to_src_chain` function in `middleware/contracts/asset-bridge/src/handle_revert.rs:16-96` initiates a refund request to the source chain.

However, the constructed request packet payload in lines `36-40` is missing important parameters that are required by the `AssetBridge` Solidity and NEAR contracts. Specifically, the sender address, source token address, and source token amount, specified in the `DepositData` and `ExecuteInfoArgs` structs in `evm/contracts/interfaces/IAssetBridge.sol:63-68` and `near/contracts/asset-bridge/src/types.rs:56-61` are not included in the payload. As a result, the refund request packet can not be decoded on the source chain, and the refund can not be processed, thus locking the user's funds on the source chain.

### Recommendation

We recommend providing the required parameters in the refund request packet payload.

**Status: Resolved**

## 15. The reentrancy lock mechanism in the NEAR `AssetBridge` contract can be abused to grief the contract

**Severity: Major**

The `ft_on_transfer` function in the NEAR `AssetBridge` contract employs a reentrancy lock mechanism to prevent calling the same function in between callbacks, which are not executed immediately but rather after 1 or 2 blocks. This is achieved using a mutex, which is set to `true` at the beginning of the function and `false` at the very end once all callbacks have been executed. If this mutex is set to `true` at the start of the function execution, the call panics.

Specifically, in `near/contracts/asset-bridge/src/execution.rs:50-51` and `74-75`, the lock variables are set to `true`, preventing the function from being repeatedly called. Once the last callback function is executed, the locks are released.

However, this lock mechanism opens up a potential Denial-of-Service (DoS) vector, as it effectively rate-limits the contract on a per-function basis. For example, an attacker can spam with many consecutive function calls every 1-2 blocks, using as little funds as possible and thus preventing any other legitimate contract calls.

While it is evident that the use of such a lock mechanism is intended to prevent reentrancy attacks, broadly applying this mechanism to all functions and blocking the functionality for a few blocks is not a suitable solution.

### Recommendation

We recommend removing the lock mechanism and ensuring that the contract's state is not exploitable between callbacks. Specifically, ensure that the `deposit_nonce` nonce is only incremented shortly before it is used within the logged event, which is currently not the case in line `46`.

**Status: Resolved**

## 16. Native token transfers always fail due to improper balance check query

**Severity: Major**

In `evm/contracts/AssetBridge.sol:582`, calling the `executeProposalForReserveToken` function for native tokens will fail. This is because before initiating the token transfer, the contract checks its balance by calling the `balanceOf` function on the token's address.

The issue here is that the contract employs the `0xEeee`… address to represent native tokens. However, `0xEee...` is not associated with an existing contract on the Ethereum network. Consequently, this balance check operation reverts, causing native token transfers to fail.

### Recommendation

We recommend separating the balance check queries for native tokens and token addresses by deploying distinct methods for each.

**Status: Resolved**

## 17. Lowercasing case-sensitive addresses causes unexpected behavior and incorrect outbound calls

**Severity: Major**

In the CosmWasm middleware contract, a critical problem exists related to address-casing sensitivity. This issue leads to unexpected and incorrect behavior, particularly when dealing with addresses from chains that employ case-sensitive address formats, such as TRON, Solana, and Polkadot. The middleware contract keeps track of `asset-bridge` contracts for various chains, which may use different address formats.

For instance, when calling the `WhiteListAddresses` message, the supplied contract address is converted to lowercase in `middleware/contracts/asset-bridge/src/execution.rs:101`. Subsequently, the whitelisted contract is incorrect because it belongs to a different TRON contract address.

### Recommendation

We recommend not broadly converting addresses to lowercase and instead only lowercasing addresses when required and only for the specific chain (such as Cosmos SDK chains or Ethereum).

**Status: Resolved**

## 18. Unintended refund due to `TEMP_REVERT_STATE` not being cleared and incorrectly calling of `revert_inbound_to_src_chain` on minting gas tokens

**Severity: Major**

In `middleware/contracts/asset-bridge/src/handle_inbound.rs:138`, `209`, `304`, and `373`, the `TEMP_REVERT_STATE` storage is used to store temporary information for reverting or refunding tokens to the sender in case an outbound request on the destination chain fails. The issue is that the `TEMP_REVERT_STATE` storage is never appropriately cleared after the transaction is completed, causing incorrect refunds when the owner calls `mint_gas_tokens` in `middleware/contracts/asset-bridge/src/execution.rs:480`.

If the `send_reserve_outbound_request` function (called inside `mint_gas_tokens`) enters `middleware/contracts/asset-bridge/src/submit_outbound.rs:36`, it will call the `revert_inbound_to_src_chain` function to issue a refund in `middleware/contracts/asset-bridge/src/handle_revert.rs:30`, which uses the uncleared `TEMP_REVERT_STATE` storage state that hold previous values.

Consequently, an outbound request will be created using the previous data stored in `TEMP_REVERT_STATE`, causing the tokens to be incorrectly refunded and a loss of funds scenario for the contract.

### Recommendation

We recommend not calling the `revert_inbound_to_src_chain` function while minting gas tokens and clearing the `TEMP_REVERT_STATE` storage once the sudo execution is completed.

**Status: Resolved**

## 19. Callback functions may run out of gas, resulting in inconsistent states of the NEAR `asset-bridge` contract

**Severity: Major**

In several instances of the NEAR codebase in `near/contracts/asset-bridge`, callbacks are implemented to handle promise results. As no validation ensures the supplied prepaid gas is sufficient for executing all promises, some callbacks might fail to execute due to an out-of-gas error.

Firstly, the `i_send_callback` function in `near/contracts/asset-bridge/src/call_gateway.rs:81` is allocated with `5 * TGAS`. This is problematic because line `159` tries to allocate the exact amount of gas if the promise fails to refund the tokens to the user. Consequently, the refund will fail, and the reentrancy lock in line `168` cannot be unlocked, causing a loss of funds and future token transfers to fail.

Secondly, the `handle_itransfer_burn_callback` function in `near/contracts/asset-bridge/src/execution.rs:183` does not enforce any static gas limit before initiating the callback. Ideally, there should be at least 15 * `TGAS` reserved because the `i_send` and `i_send_callback` functions allocate gas in `near/contracts/asset-bridge/src/call_gateway.rs:72`, lines `81`, and `159`.

Thirdly, the `handle_ireceive_callback` function in `near/contracts/asset-bridge/src/execution.rs:342`, lines `462` and `525` does not enforce sufficient gas limit. The former does not enforce any gas limit, while the latter only allocates 5 * `TGAS`. This is insufficient because a total of 10 * `TGAS` needs to be allocated in lines `568` and `576`, causing the transaction to fail due to an out-of-gas error and not reverting the reentrancy lock.

Fourthly, the `handle_ireceive_ft_get_balance_callback` function in line `364` does not enforce any static gas limit. Ideally, there should be at least 15 * `TGAS` allocated due to lines `504`, `568`, and `576`. This might cause the reentrancy lock to fail to revert, or cause the event to fail to emit by the `handle_event_by_tx_type` function.

Lastly, the `handle_unstake_callback` function in line `718` also does not enforce any static gas limit. This might cause the liquidity token redemption or refund to fail, but the user's liquidity tokens are already burned, causing a loss of funds scenario.

#### Recommendation

We recommend attaching sufficient static gas for the functions listed above. If nested promises are implemented inside a callback, the callback must ensure sufficient gas is attached to it for all the nested promises to execute successfully.

To ensure the allocated gas is sufficient, consider estimating the gas costs with automated tests or SDK tools. If there are any code changes after a gas estimation, the gas cost must be estimated again.

Additionally, since unused gas will be refunded, consider requiring the caller to attach extra gas on top of the estimated gas cost as a buffer to prevent an out-of-gas error.

**Status: Resolved**

### 20. Incorrectly determining the available NEAR balance of the `AssetBridge` contract causes transactions to fail

**Severity: Major**

In `near/contracts/asset-bridge/src/execution.rs:437`, the `handle_ireceive_with_native_token` function checks if the contract's NEAR balance is sufficient to cover the NEAR token transfer. Specifically, the balance is retrieved by calling the `env::account_balance` function and compared with the `dst_amount` amount.

However, the current storage costs of the `AssetBridge` contract are not deducted from the contract balance, causing the remaining contract's NEAR balance to be insufficient to cover the required storage costs and the transaction to fail.

### Recommendation

We recommend subtracting the current storage costs by the `total_storage_cost` function in `near/contracts/asset-bridge/src/contract.rs:110`.

**Status: Resolved**

## 21. The amount of minted gas tokens can exceed the actual charged fee amount

**Severity: Major**

In `middleware/contracts/asset-bridge/src/handle_reserve_lp_token_transfer.rs:141`, the `handle_reserve_tokens` function increases the `BURNED_GAS_TOKENS` state by `fee_deducted_amount` computed in line `109`. The state is used to mint an equivalent amount of gas tokens using the `MintGasTokens` message in `middleware/contracts/asset-bridge/src/contract.rs:148`.

The issue occurs when the computed fee is greater than the source token amount in `middleware/contracts/asset-bridge/src/handle_reserve_lp_token_transfer.rs:128`. In this case, the source token will become zero due to the `unwrap_or_default` function. However, the `fee_deducted_amount` added to the `BURNED_GAS_TOKENS` state will be larger than `src_token_amount`, which is incorrect because the actual charged fee amount equals the source token amount, not the computed fee amount.

Consequently, the owner can mint gas tokens without having an equivalent amount of tokens in the destination chain, causing users to be unable to redeem gas tokens for underlying assets due to insufficient liquidity.

### Recommendation

We recommend increasing the `BURNED_GAS_TOKENS` by the `src_token_amount` if the `fee_deducted_amount` is larger than the `src_token_amount`.

**Status: Resolved**

## 22. Panic in handler callback causes denial of service

**Severity: Major**

In several instances of the codebase, a panic can occur during promise callbacks.

Firstly, in `near/contracts/asset-bridge/src/execution.rs:492`, `496`, and `498`, a panic will occur if the U128 serialization fails or the liquidity version of the token is not configured. Panicking here will cause the `unlock_ireceive_fn` function not to execute and revert the reentrancy lock, preventing the request from being retried. This might happen if a custom token does not return U128 or if there are misconfigurations by the resource owner. The error should be handled gracefully by calling `unlock_ireceive_fn` and short-circuiting directly so the request can be retried later.

Secondly, the `handle_asset_bridge_message_callback` function in line `641` panics if the result data cannot be parsed. In this case, the `handle_event_by_tx_type` function in line `651` will not be executed to emit the relevant event. In this case, the error should be handled gracefully by setting the `exec_flag` to `false` with an empty `exec_data` and continuing execution if the result data cannot be parsed.

### Recommendation

We recommend handling the error gracefully instead of panicking.

**Status: Resolved**

## 23. The `asset-bridge` middleware contract owner can mint unlimited tokens that were previously burned

**Severity: Minor**

The `mint_gas_tokens` function in `middleware/contracts/asset-bridge/src/execution.rs:431-491` handles the `ExecuteMsg::MintGasTokens` message, which allows the owner of the CosmWasm `asset-bridge` middleware contract to mint previously burned tokens, resembling the collected fees, on a specified chain.

The number of burned tokens is retrieved from storage by calling the `fetch_burned_gas_token_amount` function in line `445` and subsequently forwarded to the specified chain via a cross-chain request.

However, the burned gas token accounting is not updated, allowing the owner to repeatedly mint the same amount of burned tokens, effectively being able to mint an unlimited amount of protocol-owned tokens.

We classify this issue as minor because it can only be caused by the contract owner, which is a privileged role.

### Recommendation

We recommend resetting the stored value of `BURNED_GAS_TOKENS` for a given `chain_id` and `token` address to zero after the owner minted gas tokens.

**Status: Resolved**

## 24. Users lose accidentally sent native tokens when staking ERC-20 tokens

**Severity: Minor**

Users can stake native or ERC-20 tokens in the `AssetBridge` Solidity contract with the `stake` function in `evm/contracts/AssetBridge.sol:601-617` and receive an equivalent amount of liquidity tokens in return. However, if a user stakes ERC-20 tokens and at the same time also supplies native tokens, i.e., `msg.value` is non-zero, those native tokens are not accounted for and are locked in the contract until the contract owner rescues them.

### Recommendation

We recommend validating in the `stake` function that `msg.value` is zero in the case of staking ERC-20 tokens.

**Status: Resolved**

## 25. Inaccurate `BURNED_GAS_TOKENS` accounting in case of errors

**Severity: Minor**

In `middleware/contracts/asset-bridge/src/handle_reserve_lp_token_transfer.rs:136-147`, the `handle_reserve_tokens` function stores the charged fee amount in the `BURNED_GAS_TOKENS` storage map if the bridged tokens on the source chain have been burned. This allows the contract owner to re-mint those burned fee tokens later.

However, if a subsequent error occurs, for example, when calling the `fetch_and_validate_dest_contract` function in `middleware/contracts/asset-bridge/src/submit_outbound.rs:29-38`, a refund is issued to the user on the source chain, but the burned gas token amount is not corrected by deducting the fees. As a result, the gas token accounting is inaccurate and allows the owner to mint more tokens than the actual accumulated fees.

### Recommendation

We recommend updating the `BURNED_GAS_TOKENS` storage map only if the cross-chain request was successfully executed.

**Status: Resolved**

## 26. `AssetBridge` Solidity contract is unable to transfer certain ERC-20 tokens

**Severity: Minor**

The `safeTransferFrom` utility function in `evm/contracts/AssetBridge.sol:278-286` is used by the `AssetBridge` contract to handle ERC-20 token transfers. The actual token transfers are performed by calling the `transfer` and `transferFrom` ERC-20 functions in lines `280` and `283`, respectively. Subsequently, the returned `bool` value is checked to be `true` and reverts otherwise.

However, some ERC-20 tokens do not return a `bool` value from their `transfer` and `transferFrom` functions. Instead, they may revert in case of an error. An example of such a token is USDT on Ethereum Mainnet.

Consequently, the `safeTransferFrom` function reverts, even though the transfer may have succeeded. This unnecessarily restricts the token usage with the `AssetBridge` contract.

### Recommendation

We recommend using OpenZeppelin's `SafeERC20` library to ensure consistent handling of ERC-20 token transfers.

**Status: Resolved**

## 27. `AssetBridge` Solidity contact is not compatible with ERC-20 tokens that charge a transfer fee

**Severity: Minor**

In `evm/contracts/AssetBridge.sol:283`, the `safeTransferFrom` utility function transfers the specified token amount from the sender address (`from`) to the recipient (`to`) by calling the `transferFrom` function of the token contract. For example, when bridging a token to a destination chain via the `transferToken` function, the funds are transferred from the caller to the `AssetBridge` contract and escrowed there.

However, ERC-20 tokens may make specific customizations to their contracts. For instance, tokens opt-in to charging a fee on every token transfer via the `transfer` or `transferFrom` functions.

Consequently, the `AssetBridge` contract incorrectly assumes that the received token amount equals the transferred token amount, leading to underfunded token transfers.

### Recommendation

We recommend carefully vetting new ERC-20 tokens before adding them to the list of allowed tokens. Additionally, we recommend adapting the `safeTransferFrom` function to compare the token balance before and after the token transfer and return the actual transferred amount. This returned amount can then be used for further processing.

**Status: Acknowledged**

## 28. Incomplete pause functionality in the codebase

**Severity: Minor**

The `AssetBridge` Solidity contract employs a pause mechanism in case of an emergency, which allows a permissioned address to pause and unpause critical functionality. However, only the `iReceive` function implemented in `evm/contracts/AssetBridge.sol:477-575` is pausable. Ideally, all entry points in the Solidity contract should be pausable with the `whenNotPaused` modifier, which are `transferToken`, `transferTokenWithInstruction`, `swapAndTransferToken`, `swapAndTransferTokenWithInstruction`, `stake`, and `unstake` functions.

This issue also occurs in the NEAR `asset-bridge` contract's `unstake` function in `near/contracts/asset-bridge/src/execution.rs:697`. Ideally, the `when_not_paused` function should be implemented to prevent users from unstaking liquidity tokens when the contract is paused.

Besides that, a pauser role is defined in `near/contracts/lp-token/src/lib.rs:102`, but there are no pausing mechanisms implemented in the `lp-token` contract.

Lastly, the `init_pausable` function is called in `middleware/contracts/asset-bridge/src/contract.rs:69`, which sets the `PAUSED` state as `true`, but the contract does not check and enforce the state across the codebase.

### Recommendation

We recommend applying pausable validations in the `asset-bridge` contracts, completing the pausable implementation in the `lp-token` contract, and enforcing the `PAUSED` state in the middleware contract.

**Status: Resolved**

## 29. Potential precision loss for values larger than 2^53-1

**Severity: Minor**

In `near/contracts/asset-bridge/src/contract.rs:110`, the `total_storage_cost` function returns the storage cost value as `u128`.

This is problematic because Javascript can only support integers up to `2^53-1` value, causing a loss of precision if the provided values are larger than that range. Specifically, the excess values will be truncated, causing the final value to differ from the supplied value.

By default, return values are serialized in JSON unless explicitly modified. This means that the value of `u128` will be serialized as numbers in JSON, which causes a loss of precision if it is larger than `2^53-1`.

This issue also exists in `near/contracts/lp-token/src/lib.rs:211` and `near/contracts/token/src/lib.rs:129`.

Consequently, the returned total storage cost value will be incorrect.

### Recommendation

We recommend modifying the implementation to use `U128` from `near_sdk::json_types` so the integers are serialized as strings instead of numbers, ensuring guaranteed precision.

**Status: Resolved**

## 30. NEAR contracts' initialization process can be front-run

**Severity: Minor**

In `near/contracts/asset-bridge/src/contract.rs:55`, the `new` function used to instantiate the NEAR `asset-bridge` contract does not implement any access control. This means anyone can initialize the contract with any values after the contract is deployed.

This issue also exists in `near/contracts/lp-token/src/lib.rs:60` and `near/contracts/token/src/lib.rs:46`.

We classify this issue as minor because the deployer can deploy the contract into another account or implement a state migration to modify the values correctly.

### Recommendation

We recommend adding the `#[private]` annotation to the `asset-bridge`, `lp-token`, and `token` contract initialization phase.

**Status: Resolved**

## 31. The recipient will receive the contract's NEAR balance instead of the specified amount

**Severity: Minor**

In `near/contracts/asset-bridge/src/execution.rs:798`, the `rescue_funds` function transfers all the contract's NEAR balance to the recipient without respecting the provided `amount` parameter. This might confuse the admin in a way that the recipient will receive the specified `amount` of NEAR tokens, which is incorrect because the whole balance will be sent instead.

We classify this issue as minor because only the admin can call the `rescue_funds` function.

### Recommendation

We recommend sending the recipient the provided `amount` of NEAR tokens instead of the available balance.

**Status: Resolved**

## 32. Incorrect token price decimal scaling may result in incorrect fee calculation

**Severity: Minor**

In `middleware/contracts/fee-manager/src/queries.rs:129`, the `get_symbol_price` function retrieves the manually set price of the specified token and falls back to Router Chain's price feed if the price is not manually set. The middleware contract operates internally on 9 decimals, and thus, token prices have to be scaled accordingly.

Router Chain's price feed potentially returns token prices with various decimals. However, in line `141`, the token price is always assumed to be in 18 decimals and is downscaled to 9 decimals. This leads to an incorrectly scaled token price being returned and potentially less than the expected 9 decimals.

Consequently, the fee calculation for the `GetFee` query message in line `217` will return an incorrect fee.

### Recommendation

We recommend scaling the token price to 9 decimals based on the actual decimals of the token price returned from Router Chain's price feed.

**Status: Resolved**

## 33. Zero address cannot be used to convert native funds

**Severity: Minor**

In `evm/contracts/AssetBridge.sol:118`, the `isNative` function determines whether the token is native by checking whether it is the zero address or `ETH_ADDRESS`. It is used within many instances where a cross-chain request is to be initiated. For example, the `transferToken` function calls the `isNative` function in line `302` after performing validation with the `tokenAndAmountValidation` function in line `301`.

The issue is that the `tokenAndAmountValidation` function checks whether the token is whitelisted in line `267`, but the `setWhiteListTokenMulti` in line `172` does not allow the zero address to be whitelisted. This prevents users from specifying the zero address for conversion into wrapped native funds.

### Recommendation

We recommend modifying the `isNative` function only to consider `ETH_ADDRESS` as native tokens, as the zero address is mostly used to check for a default address in Solidity.

**Status: Resolved**

## 34. Incorrect token refunds to the recipient instead of the sender

**Severity: Minor**

In `near/contracts/asset-bridge/src/execution.rs:781-793`, the underlying asset is minted to the recipient if the liquidity token redemption fails. This is incorrect because the refund should be returned to the sender, not the recipient. This might cause issues when the sender expects the funds to be refunded to their account or if the recipient is an immutable smart contract that cannot handle such edge cases.

### Recommendation

We recommend minting the tokens to the original sender as part of the refund process.

**Status: Resolved**

## 35. Incorrect sender assignment in NEAR `asset-bridge` contract

**Severity: Minor**

In `near/contracts/asset-bridge/src/utils.rs:518` and line `549`, the `get_itransfer_packet` and `get_itransfer_with_message_packet` functions set the `sender_token` to the caller of the function with `env::predecessor_account_id()`. This is incorrect because if the `ft_on_transfer` function is called from a fungible token contract, the `sender_token` will incorrectly be the fungible token contract address instead of the user. Consequently, the depositor address included in the packet will be incorrect.

### Recommendation

We recommend determining the `sender_token` using `sender_id` instead.

**Status: Resolved**

## 36. Lack of entry point to remove outdated storage entries

**Severity: Minor**

In `evm/contracts/AssetBridge.sol:154`, the `setLiquidityPoolMulti` function updates the `_lpToContract` and `_contractToLP` mappings for the liquidity token and the underlying asset. As there are no entry points for the owner to remove the old storage, updating the underlying asset to use a new liquidity token in `_contractToLP` will still reflect the old liquidity token as valid in `_lpToContract`.

This issue also affects the `set_chain_bytes_info` function in `middleware/contracts/asset-bridge/src/execution.rs:226-227`. If there is an update for an existing chain identifier to point to new chain bytes in `CHAIN_INFO_TO_BYTES_MAPPING`, the existing chain bytes in `CHAIN_BYTES_INFO` will still point to the old chain identifier, which is incorrect.

Additionally, the `set_token_symbols` function in `middleware/contracts/fee-manager/src/execution.rs:483-492` also suffers the same issue. If there is an update for an existing (symbol, chain identifier) to point to a new token address in `SYMBOL_TO_TOKEN`, the existing key (old token address, chain identifier) in `TOKEN_SYMBOLS` will still be valid.

### Recommendation

We recommend implementing an entry point for the owner to remove outdated states from the storage.

**Status: Partially Resolved**

## 37. Users might unintentionally lose fees if the gateway contract is not updated

**Severity: Minor**

In `evm/contracts/AssetBridge.sol:208-211`, the `routerBridge` address is encoded with the packet before dispatching it to the gateway contract. The issue is that the router bridge is not initialized during the contract initialization and will default to an empty string if called. If the `iSend` function is called before the resource owner updates the router bridge address, the transaction will fail after dispatch, causing the user-sent fees to be wasted.

This issue also affects the `get_request_packet` function in `near/contracts/asset-bridge/src/utils.rs:570`.

### Recommendation

We recommend validating that the router bridge address is not an empty string before encoding the packet to prevent fees from getting wasted.

**Status: Resolved**

## 38. Potential out-of-gas error due to unbounded query iterations

**Severity: Minor**

In `middleware/contracts/asset-bridge/src/contract.rs:202` and `204`, the `fetch_all_white_listed_contracts` and `all_pause_info` functions perform an unbounded iteration over the `WHITELISTED_ADDRESSES` and `PAUSE` storage to retrieve all the values. If there are too many storage entries to process, the query will fail due to an out-of-gas error.

### Recommendation

We recommend implementing a pagination mechanism to support batch queries.

**Status: Resolved**

## 39. Missing liquidity token configuration results in a failed refund request and the inability to retry the request

**Severity: Minor**

The `handle_ireceive_ft_get_balance_callback` function in `near/contracts/asset-bridge/src/execution.rs:473-545` is called as a callback to handle the fungible token balance query result. If the asset bridge contract's available fungible token balance is sufficient to cover the requested transfer amount, the funds are transferred to the recipient. Otherwise, an equivalent amount of liquidity tokens is minted and transferred to the recipient.

If the bridged token has no corresponding LP token configured, the `require!` assertions in line `496` and lines `498-501` will panic and abort the callback execution.

However, the request was previously marked as executed by calling the `set_execute_revert_record` function in lines `312-316` of the `i_receive` function. As this state change happened in a previous transaction, the reverting callback does not revert the request's status, resulting in the inability to retry the failed request.

On a subsequent retry attempt, the `i_receive` function will error in lines `307-311` as the request has already been marked as executed.

### Recommendation

We recommend replacing the `require!` assertions with `if` statements and reverting the state changes accordingly.

**Status: Resolved**

## 40. Usage of similar storage prefixes might cause collision

**Severity: Minor**

In `near/contracts/asset-bridge/src/contract.rs:64` and `69`, the `lp_to_token` and `execute_lock` maps use the same storage prefix. Although the unordered map structures differ, preventing a storage collision under current conditions, the risk of a collision exists if a new map with the identical storage prefix is introduced in future state migrations.

### Recommendation

We recommend distinguishing the storage prefixes to prevent storage collision.

**Status: Resolved**

## 41. Failed cross-chain request fees are borne by the `middleware` contract

**Severity: Informational**

In `middleware/contracts/asset-bridge/src/execution.rs:493`, the `execute_failed_request` function allows the owner to retry a failed cross-chain request. However, the [request fees are borne by the `middleware` contract] instead of the request sender. If the request repeatedly fails and cannot succeed, the `middleware` contract will need to pay the fees for it.

#### Recommendation

We recommend letting users pay for the cross-chain request fees.

**Status: Acknowledged**

### 42. Potential version mismatch due to hardcoded value

**Severity: Informational**

In `evm/contracts/AssetBridge.sol:210`, when utilizing the `iSend` function to send a cross-chain request through the gateway contract, the version is currently hardcoded as 1. Given that the gateway contract interface provides the `currentVersion` function, replacing the hardcoded version with `currentVersion` is advisable to ensure that the request version aligns with the gateway contract.

#### Recommendation

We recommend using the `currentVersion` function instead of hardcoding the request's version.

**Status: Acknowledged**

### 43. Inconsistency in nonce configuration for `AssetBridge` contract

**Severity: Informational**

In `evm/contracts/AssetBridge.sol:103`, the `constructor` function allows the contract instantiator to configure the `depositNonce` value based on the `startNonce`. This is inconsistent with the NEAR `asset-bridge` contract in `near/contracts/asset-bridge/src/contract.rs:60` as the deposit nonce starts as zero.

#### Recommendation

We recommend removing the `startNonce` parameter and instantiating `depositNonce` as zero for consistency.

**Status: Resolved**

## 44. Incorrect vector length validation in `set_liquidity_pool`

**Severity: Informational**

In `near/contracts/asset-bridge/src/setters.rs:124-128`, the `set_liquidity_pool` function incorrectly uses the AND (&&) operator to ensure the lengths of `token_addresses`, `lp_addresses`, and `should_reset` vectors are equal. However, the validation will not work as intended because if one of the conditions is `false`, the error will not be raised if the other condition is `true`.

Consequently, the transaction will fail due to an out-of-bounds error.

### Recommendation

We recommend updating the conditional check to use the OR (||) operator.

**Status: Resolved**

## 45. `nonReentrant` modifiers are not added for external calls

**Severity: Informational**

In `evm/contracts/AssetBridge.sol:388` and line `438`, the `swapAndTransferToken` and `swapAndTransferTokenWithInstruction` functions call the DexSpan contract to perform token swaps. As the `nonReentrant` modifier is not implemented in both functions, reentrancy vulnerabilities in the DexSpan contract can affect the `AssetBridge` contract.

### Recommendation

We recommend implementing the `nonReentrant` modifier for the `swapAndTransferToken` and `swapAndTransferTokenWithInstruction` functions.

**Status: Resolved**

## 46. Inconsistency in `srcChainId` validation

**Severity: Informational**

In `evm/contracts/AssetBridge.sol:480`, the `iReceive` function does not validate whether the `srcChainId` value equals Router Chain's ID. This is inconsistent with the `i_receive` function in `near/contracts/asset-bridge/src/execution.rs:268` as it ensures the source chain equals `router_chain_id`. If a malicious actor can control the Router bridge address on another EVM chain, this can be exploited to mint an arbitrary amount of tokens.

We classify this issue as informational due to the unlikely possibility of the attack scenario.

### Recommendation

We recommend validating the `srcChainId` equals the `ROUTER_CHAIN_ID` defined in `evm/contracts/AssetBridge.sol:27`.

**Status: Resolved**

## 47. Redundant `is_native` parameter

**Severity: Informational**

In `near/contracts/asset-bridge/src/execution.rs:165`, the `is_native` boolean is combined with the source token address validation to determine whether the token is a native asset. This is redundant as the `is_native` parameter can be removed and checked directly with the source token address.

### Recommendation

We recommend removing the `is_native` boolean parameter.

**Status: Resolved**

## 48. Unneeded payable annotation

**Severity: Informational**

In `near/contracts/asset-bridge/src/execution.rs:253`, the `i_receive` function implements the `#[payable]` annotation, which means NEAR can be attached as a deposit to the function. This is unneeded because the `execute_handler_calls` function in the `gateway` contract does not attach any deposit when calling the asset-bridge contract.

### Recommendation

We recommend removing the `#[payable]` annotation.

**Status: Resolved**

## 49. Incorrect events emitted

**Severity: Informational**

In several instances of the codebase, incorrect events are emitted:

- `middleware/contracts/asset-bridge/src/execution.rs:412`: `SetFeePayer` should be `SetRouterChainId`.

- `middleware/contracts/asset-bridge/src/execution.rs:489`: `SetStGasLimit` should be `MintGasTokens`.
- `middleware/contracts/asset-bridge/src/execution.rs:508`: `SetStGasLimit` should be `ExecuteFailedReq`.

#### Recommendation

We recommend updating the event attributes mentioned above.

**Status: Resolved**

### 50. Incomplete resource setter role mechanism

**Severity: Informational**

In `middleware/contracts/asset-bridge/src/contract.rs:70`, the `set_up_role` function is called to create the resource setter role. However, there is no entry point for the admin to call `grant_role` to grant privileges to members. While the role is meant to update configurations in the codebase, it is not enforced, as it is all delegated to the owner.

#### Recommendation

We recommend completing the role mechanism so the resource setter updates the configuration instead of the owner.

**Status: Resolved**

### 51. Token decimals are stored in both the asset bridge middleware and fee manager contracts, leading to potential inconsistencies

**Severity: Informational**

Token decimals are stored in two different places: in the middleware contract by calling the `set_token_decimals` function in `middleware/contracts/asset-bridge/src/execution.rs:121-154` as well as in the fee manager contract via the `set_token_decimals` function in `middleware/contracts/fee-manager/src/execution.rs:513-543`.

However, only the token decimals stored in the middleware contract are retrieved by the `fetch_token_decimals` function in `middleware/contracts/asset-bridge/src/queries.rs:103-109`.

As a result, the two token decimal storages may diverge if the token decimals are updated directly in the fee manager contract but not in the middleware, which may cause unexpected behavior.

### Recommendation

We recommend consolidating the token decimals storage to a single location.

**Status: Acknowledged**

## 52. Incorrect errors in NEAR `asset-bridge` contract

**Severity: Informational**

In several instances of the codebase, incorrect errors are used:

- `near/contracts/asset-bridge/src/execution.rs:821`: `asset-forwarder` should be `asset-bridge`.
- `near/contracts/asset-bridge/src/setters.rs:59`: `set_router_middleware_base` should be `set_gateway`.
- `near/contracts/asset-bridge/src/setters.rs:209`, `214`: `grant_role` should be `revoke_role`.
- `near/contracts/asset-bridge/src/utils.rs:119`, `129`, `135`, `145`, `153`, `159`, `169`, `177`, `185`, `193`: `i_transfer` should be `itransfer_token_with_message`.
- `near/contracts/asset-bridge/src/utils.rs:237, 243, i_transfer` should be `decode_ireceive_deposit`.
- `near/contracts/asset-bridge/src/utils.rs:402, 408: i_transfer` should be `decode_ireceive_execute`.
- `near/contracts/asset-bridge/src/utils.rs:455, 461, 479`: `i_transfer` should be `decode_ireceive_execute_with_message`.

### Recommendation

We recommend correcting the errors mentioned above.

**Status: Resolved**

## 53. Contracts should implement a two-step ownership transfer

**Severity: Informational**

The contracts within the scope of this audit allow the current owner to execute a one-step ownership transfer. While this is common practice, it presents a risk for the ownership of the contract to become lost if the owner transfers ownership to the incorrect address. A two-step ownership transfer will allow the current owner to propose a new owner, and then the account that is proposed as the new owner may call a function that will allow them to claim ownership and actually execute the config update.

### Recommendation

We recommend implementing a two-step ownership transfer. The flow can be as follows:

1. The current owner proposes a new owner address that is validated and converted to lowercase.
2. The new owner account claims ownership, which applies the configuration changes.

**Status: Acknowledged**