Audit Report

Router EVM and NEAR Gateway Contracts and WASM Bindings

v1.0

May 29, 2024

Table of Contents

License

Disclaimer

THE CONTENT OF THIS AUDIT REPORT IS PROVIDED “AS IS”, WITHOUT REPRESENTATIONS AND WARRANTIES OF ANY KIND.

THE AUTHOR AND HIS EMPLOYER DISCLAIM ANY LIABILITY FOR DAMAGE ARISING OUT OF, OR IN CONNECTION WITH, THIS AUDIT REPORT.

THIS AUDIT REPORT WAS PREPARED EXCLUSIVELY FOR AND IN THE INTEREST OF THE CLIENT AND SHALL NOT CONSTRUE ANY LEGAL RELATIONSHIP TOWARDS THIRD PARTIES. IN PARTICULAR, THE AUTHOR AND HIS EMPLOYER UNDERTAKE NO LIABILITY OR RESPONSIBILITY TOWARDS THIRD PARTIES AND PROVIDE NO WARRANTIES REGARDING THE FACTUAL ACCURACY OR COMPLETENESS OF THE AUDIT REPORT.

FOR THE AVOIDANCE OF DOUBT, NOTHING CONTAINED IN THIS AUDIT REPORT SHALL BE CONSTRUED TO IMPOSE ADDITIONAL OBLIGATIONS ON COMPANY, INCLUDING WITHOUT LIMITATION WARRANTIES OR LIABILITIES.

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's EVM and NEAR gateway contracts and WASM bindings.

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 targets:

Repository	https://github.com/router-protocol/router-gateway-contracts
Commit	`e863130d284eb52c8c1a8fe8859f5495ee448853`
Scope	Only the contracts in the `evm/*` and `near/gateway-upgradeable/*` directories were in the scope of the audit.
Identifier	In this report, all paths pointing to this repository are prefixed with `gateway-contracts:`
Fixes verified at commit	`6ecfda2fa2fecfb105d7f5cb395fa643d85a5435` 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.

Repository	https://github.com/router-protocol/router-wasm-bindings
Commit	`103ef1b704e8aee3d2b25f73b7cd4fc5ed834840`
Scope	Only the files in the `packages/bindings/*` directory were in the scope of the audit.
Identifier	In this report, all paths pointing to this repository are prefixed with `wasm-bindings:`
Fixes verified at commit	`c48ccfcf22b019262a314e1c2ac5c7f75c0caa77` 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-High**	The NEAR gateway contract locks and unlocks the currently processed cross-chain requests by utilizing the `is_i_receive_locked` map in many places. In many instances, following unlocking or locking, a system panic is intentionally triggered to stop the current execution. In those situations, reverting the lock state in the same execution context is not necessary, as the current state is rolled back automatically. However, if asynchronous callback functions are used, it is of utmost importance to revert any previous state changes in case of an error to ensure that the state is consistent across components. Nonetheless, we have observed many instances where the state of the locks and unlocks is manually reset in case of an error despite the automatic rollback by NEAR. This unnecessarily increases the code complexity and leads to confusion.
Code readability and clarity	**Medium**	The codebase contains commented code and TODO comments.
Level of documentation	**Medium**	Documentation is available at https://router-chain-docs.vercel.app/develop/message-transfer-via-crosstalk/key-concepts/high-level-architecture/.
Test coverage	**Medium**	-

Summary of Findings

No	Description	Severity	Status
1	Incorrect `isReadCall` implementation allows infinite token mints	Critical	Resolved
2	Duplicate `IReceiveEvent` event nonces in the NEAR gateway contract resulting in stuck cross-chain requests	Critical	Resolved
3	`ROUTE` tokens are not minted for invalid requests, causing a loss of funds	Critical	Resolved
4	ASM contract state is committed when token mint fails	Major	Acknowledged
5	Packet loss during `iReceive` execution due to `updateValset` transaction front-running	Major	Acknowledged
6	Validator set supermajority threshold discrepancy between the gateway contracts and Router Chain	Major	Resolved
7	Incomplete state rollback for failures in minting the `ROUTE` token	Major	Resolved
8	Execution status is incorrectly set to success when the handler address cannot be parsed	Major	Resolved
9	State rollbacks are not implemented correctly, preventing the failed packet from being retried	Major	Resolved
10	The reentrancy lock mechanism in the NEAR `GatewayUpgradeable` contract can be abused to grief the contract	Major	Resolved
11	Potential precision loss for values larger than `2^53-1`	Minor	Resolved
12	NEAR gateway contract does not handle read calls	Minor	Acknowledged
13	Improper gateway contract initialization protection	Minor	Resolved
14	Missing initialization of the inherited `ReentrancyGuardUpgradeable` contract	Minor	Resolved
15	NEAR gateway contract incorrectly uses `ADMIN_ROLE` instead of `PAUSER_ROLE` for pausing and unpausing	Minor	Resolved

16	EVM gateway contract can receive native token funds	Minor	Resolved
17	Default state initialization is incorrectly implemented	Minor	Resolved
18	Inconsistency between NEAR and Solidity gateway contract implementation	Minor	Resolved
19	Duplicate validators can be configured	Minor	Acknowledged
20	`iAck` cross-chain requests do not mark the `ack_request_identifier` nonce as executed in the NEAR gateway contract	Minor	Resolved
21	No upper limit when setting bridge fees	Informational	Resolved
22	Inconsistent chain type codes between Router Chain and WASM bindings chain codes	Informational	Resolved
23	The use of Solidity's `transfer` function might cause fee withdrawals to fail	Informational	Resolved
24	Usage of deprecated `_setupRole` function	Informational	Resolved
25	Unnecessary state rollbacks implemented	Informational	Resolved
26	Inconsistent function signature when verifying cross-chain requests	Informational	Resolved
27	Unused imports in the Solidity gateway contract	Informational	Resolved
28	Updating bridge fees does not emit events	Informational	Resolved
29	Redundant destination chain ID validation	Informational	Resolved
30	Out-of-bounds error if the signature length does not match the validator length	Informational	Acknowledged
31	Errors when parsing ASM address are ignored	Informational	Resolved
32	Signature parsing errors are not propagated correctly	Informational	Resolved
33	Lack of role-based access controls for the pausing mechanism	Informational	Resolved

Detailed Findings

1. Incorrect `isReadCall` implementation allows infinite token mints

Severity: Critical

Decentralized applications utilizing Router Chain to facilitate cross-chain contract calls can query a contract on a destination chain in a read-only manner, i.e., without performing any write operations on the destination chain. This is achieved by setting the `isReadCall` value of the `iSend`'s request metadata parameter's `requestMetadata` to `true`.

On the receiving destination chain, the EVM gateway contract handles the cross-chain request in the `iReceive` function. The target contract, specified in `requestPayload.handlerAddress`, is then called with the provided data, and depending on the specified `isReadCall` value, the call is limited to a read-only call to prevent state changes on the destination chain.

However, Solidity's low-level `call` function is incorrectly used to call an arbitrary function on the target contract address in `gateway-contracts:evm/contracts/GatewayUpgradeable.sol:427`. This means state changes are not prevented, allowing the target contract to perform write operations on the destination chain even if `isReadCall` was set to `true` by the caller.

As the gateway contract calls the `handler` contract address with the specified request payload (i.e., `RequestPayload`), authentication checks that ensure the caller is the gateway contract address can be bypassed.

Consequently, an attacker can create requests where they specify the `handler` address as the `AssetVault` contract with the `request.payload` as the `handleWithdraw` function, allowing them to mint an infinite amount of `ROUTE` tokens, as seen in `gateway-contracts:evm/contracts/AssetVault.sol:33`.

Recommendation

We recommend using Solidity's `staticcall` function instead of `call` in line `427` to enforce read-only cross-chain contract calls when `isReadCall` is set to `true`.

Status: Resolved

2. Duplicate IReceiveEvent event nonces in the NEAR gateway contract resulting in stuck cross-chain requests

Severity: Critical

The i_receive function in the NEAR gateway contract, handling relayed iReceive messages, mints ROUTE tokens if the route_amount specified in the RequestPayload parameter is non-zero. ROUTE tokens are minted by asynchronously calling the mint function from the RouterToken contract in gateway-contracts:near/gateway-upgradeable/src/lib.rs:612-620.

Subsequently, once the execution of the mint function is finished, the handle_router_token_mint_callback callback is called to handle the PromiseResult.

If minting ROUTE tokens succeeded, i.e., in the PromiseResult::Successful case, the dApp contract specified in RequestPayload.handler_address is called. In case the handler's NEAR address can not be parsed, the IReceiveEvent is emitted, and the execution finishes.

The event_nonce used for the event is retrieved from storage via self.event_nonce in gateway-contracts:near/gateway-upgradeable/src/lib.rs:815. As event nonces are expected to be unique and sequential, the event_nonce has already been incremented in the i_receive function in line 599.

However, in NEAR, callbacks are called in subsequent blocks, and consecutive i_receive calls may increase the event_nonce multiple times before the nonce is used in an event. Consequently, handle_router_token_mint_callback callbacks executed in later blocks potentially retrieve the same event_nonce from storage. This leads to duplicated nonces, causing orchestrators to be unable to attest to the conflicting events in Router Chain, ultimately resulting in stuck cross-chain requests.

Recommendation

Instead of incrementing the event_nonce in the i_receive function in line 599, we recommend incrementing the nonce in the handle_router_token_mint_callback function shortly before consuming it in the IReceiveEvent event.

Additionally, instead of supplying the self.event_nonce as the event_nonce parameter to the execute_handler_calls function in lines 643, 758, and 841, the event nonce should be retrieved from storage inside the execute_handler_calls function call and incremented there instead of passing it as a parameter.

Implementing the above recommendations will also mitigate the following potential issues:

1. The unused but previously incremented event_nonce in the PromiseResult::Failed case when minting ROUTE tokens fails leads to skipping

event nonces and results in attestation issues within Router Chain as the nonce is supposed to be sequential without gaps.

2. If decoding the `result` in the `handle_execute_handler_calls_callback` function in `gateway-contracts:near/gateway-upgradeable/src/lib.rs:925` fails, the previously incremented `event_nonce` remains unused and leads to gaps in the emitted event nonces and thus attestation issues.

Status: Resolved

3. `ROUTE` tokens are not minted for invalid requests, causing a loss of funds

Severity: Critical

In `gateway-contracts:near/gateway-upgradeable/src/lib.rs:689-714`, an `IReceiveEvent` event will be emitted as a failure when the ASM contract returns the request as invalid. In this case, the `ROUTE` tokens are not minted to the user. This is incorrect because the `ROUTE` tokens should be minted even if the `is_valid_request` value is `false`. Consequently, the route recipient will not receive the expected route amount, causing a loss of funds.

Recommendation

We recommend modifying the implementation to ensure `ROUTE` tokens are minted if the `is_valid_request` value is `false`.

Status: Resolved

4. ASM contract state is committed when token mint fails

Severity: Major

In `gateway-contracts:near/gateway-upgradeable/src/lib.rs:847-857`, the `handle_router_token_mint_callback` function reverts the state if minting `ROUTE` tokens fails. Ideally, a failed token mint should revert all committed states, including state changes in the ASM contract.

However, this is not the case. If the mint transaction is called in lines `725-733` after the ASM contract transaction succeeds, the implemented state rollbacks in lines `847-857` will not include the ASM contract. Consequently, the state changes made in the ASM contract will not revert, potentially preventing the packets from being replayed.

For example, assume an ASM contract implements a replay attack defense that will reject the packet when it is replayed again. Since the state is already committed in the first call, subsequent replays will cause the transaction to fail, preventing the token mint from working as intended.

Recommendation

We recommend modifying the implementation so the ASM contract state is reverted if the ROUTE token minting fails.

Status: Acknowledged

The client states that the ASM should not do anything when executing the data validation check. In other words, the ASM should just validate the incoming request but not mark it as processed. If they want to do that sort of state update, then they can do it from the handler execution.

5. Packet loss during iReceive execution due to updateValset transaction front-running

Severity: Major

In the context of the Gateway contract, there is an issue concerning packet loss during the execution of the iReceive method. This problem arises because the relayer calls the iReceive method, which requires the submission of signatures from the current validator sets. These signatures are then matched against the stateLastValsetCheckpoint in gateway-contracts:evm/contracts/GatewayUpgradeable.sol:165.

However, there is a potential edge case to consider. Since all unbonding validators must provide attestation on ISendEvents, there could be a situation where the previous valset has voted on ISendEvents on the source chain. These signatures are valid, but an updateValset transaction is proposed before processing the iReceive function.

To illustrate, imagine a case where a user's ROUTE tokens are burned on the source chain, but the packet is lost on the destination chain because iReceive fails due to the updated valset. In this scenario, if the packet on the target chain is in the READY_TO_EXECUTE or BLOCKED state, it could be lost and never processed. This situation breaks the interoperability guarantee that packets will always reach the destination chain.

Recommendation

We recommend implementing the mechanism to submit lost packets asynchronously, either through API or by opening a dispute.

Status: Acknowledged

The client states that the probability of changing the valset by more than 33.34% in a single valset update is low. If the valset is changing by less than 30 %, then it will not cause any issues. The client also states that they are planning to add one new independent flow to provide new signatures to handle this scenario.

6. Validator set supermajority threshold discrepancy between the gateway contracts and Router Chain

Severity: Major

Router Chain determines the supermajority of received attestation votes from all validators (i.e., orchestrators) as 66% + 1 of the total voting power, as defined in Router Chain's genesis file. The supermajority of orchestrators is required to sign and attest to the validity of cross-chain requests before they can be executed on the destination chain by submitting their signatures.

However, the EVM and NEAR gateway contracts both define the threshold at which the supermajority is reached as 65% + 1 of the total voting power, as defined in the `CONSTANT_POWER_THRESHOLD` constant in `gateway-contracts:evm/contracts/Utils.sol:56` and `gateway-contracts:near/gateway-upgradeable/src/constants.rs:3`, respectively. The value of the `CONSTANT_POWER_THRESHOLD` constant, `2791728742`, represents a value of `0.65`, as verified by calculating `2791728742 / (2^32)`.

This discrepancy between the gateway contracts and Router Chain leads to incorrectly considering cross-chain requests as valid by having 65% + 1 of the total voting power of validators sign the request.

Recommendation

We recommend updating the `CONSTANT_POWER_THRESHOLD` constant in both the EVM and NEAR gateway contracts to reflect the supermajority threshold defined in Router Chain. This can be achieved by calculating the threshold as `uint256(2 * 2**32) / 3` in Solidity, resulting in a value of `2863311530`.

Status: Resolved

7. Incomplete state rollback for failures in minting the `ROUTE` token

Severity: Major

In `gateway-contracts:near/gateway-upgradeable/src/lib.rs:854`, the `handle_router_token_mint_callback` function unlocks the `is_i_receive_locked` map in case an error occurs during `ROUTE` token minting. As the state rollback is intended for the cross-chain request to be retried by the relayer at a later time, the `nonce_executed` state should be reverted as well.

Specifically, the state is committed in the `i_receive` function (see line `596`) and the `asm_request_callback` function (see line `685`) if the ASM contract validation is required. Consequently, executing the cross-chain request cannot be retried due to the validation in line `561`.

Recommendation

We recommend rolling back the `nonce_executed` state when a promise result failure occurs in the `handle_router_token_mint_callback` function.

Status: Resolved

8. Execution status is incorrectly set to success when the handler address cannot be parsed

Severity: Major

In `gateway-contracts:near/gateway-upgradeable/src/lib.rs:821`, the `exec_status` is set to `true` when the handler address cannot be parsed correctly in line `811`. This is incorrect, as Router Chain will register the execution status as success even though the handler address has not been executed.

Recommendation

We recommend modifying the `exec_status` to `false`.

Status: Resolved

9. State rollbacks are not implemented correctly, preventing the failed packet from being retried

Severity: Major

In `gateway-contracts:near/gateway-upgradeable/src/lib.rs:772` and `856`, a panic will occur if the promise result is a failure in the `asm_request_callback` and `handle_router_token_mint_callback` functions. Since a panic reverts the transaction, the state rollback that unlocks the `is_i_receive_locked` map will not be committed.

This could happen if the Router ASM reverts in case of an error in the `asm_request_callback` function, indicating the relayer should retry the request at a later time. Consequently, the `iReceive` packet cannot be retried due to the validation in line `430`.

Recommendation

We recommend returning an empty promise instead of a panic to commit the state rollback.

Status: Resolved

10. The reentrancy lock mechanism in the NEAR `GatewayUpgradeable` contract can be abused to grief the contract

Severity: Major

The `i_receive` and `i_ack` functions in the NEAR `GatewayUpgradeable` contract employ 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, specifically, the `is_i_receive_locked` and `is_i_ack_locked` storage variables, which are set to `true` at the beginning of the function and `false` at the very end once all callbacks have been executed. If a mutex is set to `true` at the start of the function execution, the call aborts and panics.

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 many consecutive cross-chain requests to the NEAR gateway contract, 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, we recommend ensuring that the `event_nonce` is only incremented shortly before it is used within the logged event to guarantee that the nonces are unique and sequential.

Status: Resolved

11. Potential precision loss for values larger than `2^53-1`

Severity: Minor

In several instances of the codebase, the `u64` variable is required to be provided as an argument:

- `powers: Vec<u64>`

    - gateway-contracts:near/gateway-upgradeable/src/lib.rs:108
- ValsetArgs.powers: Vec<u64>
    - gateway-contracts:near/gateway-upgradeable/src/lib.rs:260,262,410,974

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. Consequently, the newly initialized and used ValsetArgs powers will be incorrect.

Besides that, the total_storage_cost function in line 1488 returns as u128. 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.

Recommendation

We recommend modifying the implementation to use U64 and U128 from near_sdk::json_types so the integers are serialized as strings instead of numbers, ensuring guaranteed precision.

Status: Resolved

12. NEAR gateway contract does not handle read calls

Severity: Minor

In gateway-contracts:near/gateway-upgradeable/src/lib.rs:861, the execute_handler_calls function does not handle the scenario when the is_read_call boolean is set to true in line 499. If this is the case, a query should be dispatched to receive the response back as an acknowledgment without performing any state changes. Since the execute_handler_calls does not handle this, state changes can happen despite the packet caller intending to perform read calls only.

Recommendation

We recommend explicitly mentioning that read-calls are not enforced on the NEAR gateway contract.

Status: Acknowledged

13. Improper gateway contract initialization protection

Severity: Minor

The upgradeable EVM gateway contract GatewayUpgradeable uses the Universal Upgradeable Proxy Standard (UUPS) pattern, consisting of a minimal proxy contract and an implementation contract. The proxy contract stores the address of the implementation contract and delegates all calls to the implementation contract. This implementation contract also contains the code required for upgrading the contract, i.e., changing the implementation contract address stored in the proxy contract.

The GatewayUpgradeable contract is initialized by calling the initialize function via the proxy, implemented in gateway-contracts:evm/contracts/GatewayUpgradeable.sol:191-227. Re-initializing the contract by consecutively calling the initialize function will revert due to the initializer modifier, inherited from OpenZeppelin's Initializable contract.

However, as the initialize function is permissionless, anyone can call this function on the implementation contract itself, bypassing the proxy contract's initialization state. For example, an attacker can front-run the deployment process and call the initialize function to set the parameters to invalid values.

Fortunately, besides an incorrectly initialized implementation contract, we did not identify any immediate security harms as the contract does not contain any sensitive functions such as delegatecall or selfdestruct.

Consequently, this will complicate the deployment of pre-computed addresses, forcing the contract instantiator to deploy a new contract.

This issue is also present in gateway-contracts:near/gateway-upgradeable/src/lib.rs:105 for the NEAR gateway contract. Since the new function does not implement access controls, attackers can front-run the deployment process to initialize the contract with invalid parameters.

Recommendation

We recommend implementing access controls for both gateway contracts to ensure only a permissioned caller can call the initialize function. Additionally, OpenZeppelin's recommendation should be followed by calling the Initializable._disableInitializers function in the Solidity gateway contract's constructor and adding #[private] annotation to the NEAR gateway contract initialization phases.

Status: Resolved

14. Missing initialization of the inherited ReentrancyGuardUpgradeable contract

Severity: Minor

The GatewayUpgradeable contract inherits from OpenZeppelin's ReentrancyGuardUpgradeable contract, providing modifiers such as nonReentrant to prevent reentrancy.

However, the GatewayUpgradeable contract does not call the __ReentrancyGuard_init function in the initialize function implemented in gateway-contracts:evm/contracts/GatewayUpgradeable.sol:191-227. Consequently, the status storage variable, used to track the current reentrancy status, is not initialized to _NOT_ENTERED, equal to 1, but to the default value of 0 instead.

Nonetheless, the reentrancy protection works as expected as the nonReentrant modifier prevents reentrancy by asserting that the current status is not equal to _ENTERED, i.e., 2.

Recommendation

We recommend calling the inherited __ReentrancyGuard_init function in the initialize function to ensure the correct initialization of the ReentrancyGuardUpgradeable contract.

Status: Resolved

15. NEAR gateway contract incorrectly uses ADMIN_ROLE instead of PAUSER_ROLE for pausing and unpausing

Severity: Minor

The NEAR gateway contract GatewayUpgradeable can be paused and unpaused by calling the pause and unpause functions, implemented in gateway-contracts:near/gateway-upgradeable/src/lib.rs:1296-1306 and lines 1309-1319, respectively.

Both functions are access-controlled and can only be called by an account with the appropriate role. However, even though the contract specifies a separate PAUSER_ROLE role, this role is not used. Instead, the ADMIN_ROLE role is checked for the caller, failing to segregate permissions.

Recommendation

We recommend using the PAUSER_ROLE role for pausing and unpausing the contract.

Status: Resolved

16. EVM gateway contract can receive native token funds

Severity: Minor

The EVM gateway contract GatewayUpgradeable implements the receive function in gateway-contracts:evm/contracts/GatewayUpgradeable.sol:503, allowing the contract to receive direct native token transfers. However, native tokens are only expected to be received from a caller calling the payable setDappMetadata and iSend functions to cover the iSendDefaultFee bridge fee. All other native token transfers should be prevented to avoid accidental fund transfers.

Recommendation

We recommend removing the receive function to prevent accidental token transfers.

Status: Resolved

17. Default state initialization is incorrectly implemented

Severity: Minor

By default, NEAR's SDK allows contracts to be initialized with a default state defined in the Default trait. Normally, if there is a need to customize the initialization of the contract, a separate #[init] annotated function can be used, which takes parameters and performs custom logic. In this case, the default state initialization should be prevented to follow NEAR's best practices.

The Router's NEAR gateway contract requires such a custom initialization and thus implements a separate #[init] annotated function in gateway-contracts:near/gateway-upgradeable/src/lib.rs:105-182. However, the Default trait is still implemented in lines 79-99, which is not required and can be removed.

Consequently, if users call public functions in the gateway contract before the contract is instantiated, the default values will be used. This will be misleading as the default values, such as the empty chain identifier string in line 83, are not implemented correctly.

Recommendation

We recommend adding PanicOnDefault to the derive macro and removing the Default trait implementation.

Status: Resolved

18. Inconsistency between NEAR and Solidity gateway contract implementation

Severity: Minor

In several instances of the codebase, the implementation between NEAR and Solidity gateway contracts differs. Specifically, the same type of errors would result in different results.

Firstly, if there is an error parsing the handler address in `gateway-contracts:near/gateway-upgradeable/src/lib.rs:627`, a panic will occur directly and revert the transaction. This is inconsistent with the Solidity implementation as errors in the handler address will cause the `execFlag` to become `false` instead of revert in `gateway-contracts:evm/contracts/GatewayUpgradeable.sol:418` and line `427`. This issue is also present in `gateway-contracts:near/gateway-upgradeable/src/lib.rs:570`, `627`, and `740`.

Secondly, if there is an error parsing the execution data in `gateway-contracts:near/gateway-upgradeable/src/lib.rs:927`, a panic will occur directly and revert the transaction. This is inconsistent with the Solidity implementation because the `IReceiveEvent` event will be emitted regardless of the execution data result in `gateway-contracts:evm/contracts/GatewayUpgradeable.sol:437`.

Thirdly, if there is an error parsing the request sender address in `gateway-contracts:near/gateway-upgradeable/src/lib.rs:1137`, a panic will occur directly and revert the transaction. This is inconsistent with the Solidity implementation because the `IAckEvent` event will be emitted regardless of the execution data result in `gateway-contracts:evm/contracts/GatewayUpgradeable.sol:492`.

Recommendation

We recommend ensuring consistency across the NEAR and Solidity gateway implementations. For example, consider emitting the event as a failure instead of panic.

Status: Resolved

19. Duplicate validators can be configured

Severity: Minor

In `gateway-contracts:evm/contracts/GatewayUpgradeable.sol:193`, no validation ensures the contract instantiator does not provide duplicate validator addresses when instantiating the gateway contract.

If duplicate validator addresses are provided, the total cumulative power in `gateway-contracts:evm/contracts/libraries/ValsetUpdate.sol:11` would be inflated, bypassing the constant power threshold validation.

This issue is also present in `gateway-contracts:near/gateway-upgradeable/src/lib.rs:107` for the NEAR gateway contract.

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

Recommendation

We recommend deduping the validator addresses when instantiating the gateway contract.

Status: Acknowledged

The client states that their deployer will ensure they pass the correct validator set in the right order. Even if they made a mistake and configured an invalid validator set they can re-deploy again. Hence, the client considers this issue as a deployment precautionary practice.

20. `iAck` cross-chain requests do not mark the `ack_request_identifier` nonce as executed in the NEAR gateway contract

Severity: Minor

`iAck` cross-chain requests, enabling the dApp to receive acknowledgments for the relayed cross-chain request, are processed by the NEAR gateway contract in the `i_ack` function. However, contrary to the EVM gateway contract, the `ack_request_identifier` nonce representing the `iReceive` event nonce on the destination chain is not marked as executed in the `nonce_executed` map in `gateway-contracts:near/gateway-upgradeable/src/lib.rs:1039-1059`.

While we did not find any security-related issues due to event nonces being unique on a given chain, it is inconsistent with the EVM gateway contract's implementation and should be unified.

Recommendation

We recommend marking the `ack_request_identifier` nonce with the given `dest_chain_id` as executed in the `nonce_executed` map in lines `1199-1202` of the `handle_crosschain_ack_callback` function.

Status: Resolved

21. No upper limit when setting bridge fees

Severity: Informational

In gateway-contracts:evm/contracts/GatewayUpgradeable.sol:235, there is no upper limit validation for the setBridgeFees function. This means that the owner can set the iSendDefaultFee to a prohibitively large value.

Recommendation

We recommend enforcing a maximum cap for iSendDefaultFee.

Status: Resolved

22. Inconsistent chain type codes between Router Chain and WASM bindings chain codes

Severity: Informational

The get_chain_code function in wasm-bindings:packages/bindings/src/types.rs:88 returns the chain code for a specific ChainType enum value. For instance, the chain code for ChainType::ChainTypeEvm is 1. However, the returned chain codes do not match the definition in the multichain Router Chain module where the code for EVM-based chains is 2.

Specifically, the chain codes seem to be shifted by 1 in Router Chain's multichain module.

While the internal use of the get_chain_code function within the WASM bindings should not lead to any issues, any external use by integrating the bindings in a CosmWasm contract could lead to unexpected behavior when interacting with Router Chain.

Recommendation

We recommend updating the chain codes in the get_chain_code function to match those defined in the multichain Router Chain module.

Status: Resolved

23. The use of Solidity's transfer function might cause fee withdrawals to fail

Severity: Informational

Fees accumulated in the GatewayUpgradeable contract can be withdrawn using the withdrawFee function. The function uses Solidity's transfer function to send the contract's current native token balance to the specified `recipient` address in `gateway-contracts:evm/contracts/GatewayUpgradeable.sol:244`.

However, Solidity's `transfer` function only forwards a gas stipend of 2300 to the recipient address, leading to issues when sending to a contract. Specifically, the transfer will inevitably fail when the contract's `receive` or `payable fallback` function consumes more than the forwarded 2300 gas units.

Recommendation

We recommend using a low-level `call` to ensure funds are sent properly.

Status: Resolved

24. Usage of deprecated `_setupRole` function

Severity: Informational

In `gateway-contracts:evm/contracts/GatewayUpgradeable.sol:217`, the `_setupRole` function is called when instantiating the gateway contract. However, the `_setupRole` function is deprecated, as mentioned in the OpenZeppelin documentation.

Recommendation

We recommend using the `_grantRole` function.

Status: Resolved

25. Unnecessary state rollbacks implemented

Severity: Informational

In several instances of the NEAR gateway codebase, state rollbacks are implemented before a panic. As panics will revert to the current state, manual state rollbacks are not required. For example, the `is_i_receive_locked` map is unlocked in `gateway-contracts:near/gateway-upgradeable/src/lib.rs:442`, then a panic occurs in line `449`. In this case, the unlock is not needed as the panic will revert the transaction state.

Recommendation

We recommend removing the unnecessary state rollbacks to improve code readability and maintainability.

Status: Resolved

26. Inconsistent function signature when verifying cross-chain requests

Severity: Informational

The Additional Security Module (ASM) serves as a plugin, enabling developers to easily incorporate their custom security mechanism into their dApp without the need for significant changes to their existing code.

Router defines an interface for such ASM contracts, the `IAdditionalSecurityModule` interface with the `verifyCrossChainRequest` function for EVM-based Solidity contracts, and the `AsmContract` trait with the `verify_cross_chain_request` function for NEAR-based Rust contracts.

However, the function signatures for both functions are inconsistent. Specifically, in the Solidity interface in `gateway-contracts:evm/contracts/IAdditionalSecurityModule.sol:11-12`, the `srcChainId` function parameter is defined after the `requestSender` parameter, while in NEAR, `src_chain_id` is defined before the `request_sender` parameter in `gateway-contracts:near/gateway-upgradeable/src/external.rs:22-23`.

This inconsistency can lead to confusion when implementing an ASM contract for both EVM chains and NEAR, as the function signatures may be expected to be identical.

Recommendation

We recommend unifying the function signatures in the `IAdditionalSecurityModule` interface and the `AsmContract` trait.

Status: Resolved

27. Unused imports in the Solidity gateway contract

Severity: Informational

The `GatewayUpgradeable` contract imports OpenZeppelin's `Context` contract in `gateway-contracts:evm/contracts/GatewayUpgradeable.sol:9`. However, the `Context` contract is not used anywhere in the contract and thus can be removed.

Recommendation

We recommend removing the `Context` import from the `GatewayUpgradeable` contract to clean up the imports.

Status: Resolved

28. Updating bridge fees does not emit events

Severity: Informational

In `gateway-contracts:evm/contracts/GatewayUpgradeable.sol:235`, the `setBridgeFees` function does not emit an event after updating the bridge fee.

Emitting an event would be useful for third-party protocols and users to adjust the sent funds to match the required fee amount.

This issue is also present in `gateway-contracts:near/gateway-upgradeable/src/lib.rs:1263` for the NEAR gateway contract.

Recommendation

We recommend emitting an event after updating the fees.

Status: Resolved

29. Redundant destination chain ID validation

Severity: Informational

In `gateway-contracts:evm/contracts/GatewayUpgradeable.sol:366`, a validation exists to ensure the `chainId` equals to `requestPayload.destChainId`. This validation can be streamlined to improve code efficiency by directly using the `chainId` value in line `356`, similar to the `iAck` function in line `461`.

Recommendation

We recommend using the `chainId` when encoding the ABI in line `356`.

Status: Resolved

30. Out-of-bounds error if the signature length does not match the validator length

Severity: Informational

In `gateway-contracts:evm/contracts/SignatureUtils.sol:25`, the signature length is not validated to be equal to the total validator length. If the `checkValidatorSignatures` function is not able to access the signature value for a validator, an out-of-bounds error will occur.

This issue is also present in `gateway-contracts:near/gateway-upgradeable/src/singnature_utils.rs:53` for the NEAR gateway contract.

Recommendation

We recommend adding validation to ensure the signature length equals the validator length.

Status: Acknowledged

31. Errors when parsing ASM address are ignored

Severity: Informational

In `wasm-bindings:packages/bindings/src/types.rs:64`, the ASM address is parsed with the `unwrap_or_default` function. This means if an error occurs when parsing the ASM address, it will be ignored, and the ASM address will default into an empty string.

Recommendation

We recommend directly calling the `unwrap` function so errors are raised when parsing the ASM address.

Status: Resolved

32. Signature parsing errors are not propagated correctly

Severity: Informational

In `gateway-contracts:near/gateway-upgradeable/src/singnature_utils.rs:137`, the `unwrap` function is called directly when parsing signature bytes. If an error occurs, the function will panic, which is inconsistent with other error-handling approaches. For example, if there is an error recovering the public key in line `139`, the error is returned gracefully in line `146` instead of unwrapping directly.

Recommendation

We recommend handling the error gracefully instead of unwrapping it directly.

Status: Resolved

33. Lack of role-based access controls for the pausing mechanism

Severity: Informational

The Solidity gateway contract implements a pausing mechanism, which is in line with best practices. However, all of the administrative functions of the contract are centralized in the admin role, which goes against the principle of least privilege.

Segregating the pauser role has the additional benefit of swifter reactions in case of need when assigned to an EOA compared to the admin that might be managed by a multisig or a governance contract.

Recommendation

We recommend implementing a separate pauser role that can turn the pausing mechanism on and off, similar to the NEAR gateway contract.

Status: Resolved