Audit Report

Router Voyager Forwarder and CW Gateway

v1.2

April 30, 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 Voyager Forwarder and CW Gateway.

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/asset-forwarder-contracts
Commit	`47f8e3ce2edc333e1c063820d33812b2570e85dd`
Scope	All contracts excluding the `evm/src/dexspan/*` directory were in scope.
Identifier	In this report, all paths pointing to this repository are prefixed with `asset-forwarder-contracts:`
Fixes verified at commit	`705e8febd4bfaf7e1102eacdb5fa71517ea2ddac` 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/asset-forwarder-middleware
Commit	`18589365cb79a39a509d5958b69257de22fb5fe4`
Scope	All contracts were in scope.
Identifier	In this report, all paths pointing to this repository are prefixed with `asset-forwarder-middleware:`
Fixes verified at commit	`223fb80e7fa5bfaf1d87f6dfb7a2b5302378b0f7` 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/cosmwasm_gateway_contract
Commit	`7ad7fb087030d6c605bd5129f39e5f03b07d4f13`
Scope	All contracts were in scope.
Identifier	In this report, all paths pointing to this repository are prefixed with `cosmwasm-gateway-contract:`
Fixes verified at commit	`ede4b0a0af0ec9a3b4b2b2784b750ccfb8b1366b` 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 not deducting the required fees and missing partner fees.
Level of documentation	**Medium-High**	The client provided recorded videos and detailed documentation.
Test coverage	**Low**	Tests in the `asset-forwarder-middleware` repository are broken due to compiler errors. There are no tests in the `cosmwasm-gateway-contract` repository.

Summary of Findings

No	Description	Severity	Status
1	Incorrect reentrancy unlock causes denial of service	Critical	Resolved
2	Attackers can create deposit requests without including funds	Critical	Resolved
3	Unauthorized `ibc_channel_connect` may lead to unauthorized calls	Critical	Resolved
4	Nonce is incorrectly incremented when fungible tokens transfer fails	Critical	Resolved
5	Attackers can register and deregister their accounts for profit	Critical	Resolved
6	Lowercasing case-sensitive addresses causes unexpected behavior and loss of funds	Critical	Resolved
7	Incorrect tokens are used to account for claimable tokens, causing a loss of funds	Critical	Resolved
8	Attackers can steal funds by repeatedly withdrawing blocked funds	Critical	Resolved
9	Erroneous claimable updates lead to the loss of the forwarder's funds	Critical	Resolved
10	Extra fees can be unboundedly added to a deposit, resulting in failure to conclude the deposit	Critical	Acknowledged
11	Updating the forwarder's claimable amount during a withdrawal request uses the wrong decimal precision, resulting in the ability to steal funds	Critical	Resolved
12	Incorrect access control and entry point implementation	Critical	Resolved
13	Risk of inaccurate account ID determination	Critical	Resolved
14	Continuously failing ERC-20 token transfers in the `iReceive` function leads to stuck funds	Critical	Resolved
15	Timeout for `SendPacket::ReceivePayload` IBC packets are not handled, resulting in lost `ROUTE` tokens	Critical	Resolved

16	`HandlerExecMsg::IReceive` message is sent to the handler contract regardless of the `ASM` contract execution result	Critical	Resolved
17	Unordered IBC channel is incorrectly enforced, resulting in out-of-order IBC packets	Critical	Resolved
18	Incorrect data parsing for `HandlerExecMsg::IReceive` and `HandlerExecMsg::IAck` messages resulting in lost `ROUTE` tokens	Critical	Resolved
19	Panic in handler callback causes denial of service and loss of funds	Critical	Resolved
20	Inability to withdraw pending forwarder funds if there is no matching fund deposit	Major	Acknowledged
21	Failed `ASM` contract call or `ROUTE` token minting does not abort the whole transaction, causing partial state to be committed	Major	Acknowledged
22	The reentrancy lock mechanism in the NEAR `asset-forwarder` contract can be abused to grief the contract	Major	Resolved
23	Admin cannot withdraw native tokens from the `asset-forwarder` contract	Major	Resolved
24	Hardcoded gas limits might cause cross-chain messages to fail	Major	Resolved
25	Depositors are unable to withdraw blocked funds due to incorrect address format	Major	Resolved
26	Extra fees added to a relayed deposit cannot be withdrawn	Major	Acknowledged
27	User's `create_refund_request` can grief forwarders, preventing fund retrieval	Major	Resolved
28	The sender chain will not be notified of errors via IBC acknowledgment	Major	Resolved
29	Handling `RequestPayload` IBC packet will fail due to insufficient integer values	Major	Resolved
30	Incorrect system fee calculation when computing fee distribution	Major	Resolved
31	Lack of entry point to remove outdated storage entries	Major	Acknowledged

32	Denial of service due to unbounded processing of forwarder balances	Major	Resolved
33	Refund request fees are borne by the `middleware` contract	Major	Acknowledged
34	Extra fee token’s liquidity is updated with an incorrect value	Major	Resolved
35	Failure to decode the recipient address leads to loss of funds	Major	Acknowledged
36	Scaling decimal tokens might cause precision issues	Major	Acknowledged
37	Callback functions may run out of gas, resulting in inconsistent states of the NEAR `asset-forwarder` contract	Major	Resolved
38	Failed `iAck` messages can be replayed	Major	Resolved
39	Chain pause mechanism is not enforced	Major	Resolved
40	Potential precision loss for values larger than `2^53-1`	Minor	Resolved
41	Specifying a large `dest_amount` value could lead to funds overspending	Minor	Acknowledged
42	CosmWasm `gateway` contract can not be paused in case of an emergency	Minor	Resolved
43	The community pause mechanism can be abused to grief the Solidity `asset-forwarder` contract	Minor	Acknowledged
44	Using Solidity's `transfer` function may prevent relaying funds to the destination chain	Minor	Resolved
45	Fees are not validated to be below 100%	Minor	Resolved
46	NEAR contracts’ initialization process can be front-run	Minor	Resolved
47	Minimum and maximum pause staked amount is not validated	Minor	Resolved
48	The recipient will receive the contract’s NEAR balance instead of the specified amount	Minor	Resolved
49	Incomplete fee mechanism and system fee withdrawal	Minor	Resolved
50	Incompatibility of deposit ID integer types	Minor	Acknowledged

51	Potential out-of-gas error due to unbounded query iterations	Minor	Partially Resolved
52	`FetchBalance` query message will always return zero `ROUTE` tokens	Minor	Resolved
53	Inconsistent state in case of an error during the `fund_deposit_post_processing` function call, resulting in the depositor receiving the funds twice	Minor	Resolved
54	Incorrect errors in NEAR `asset-forwarder` contract	Informational	Resolved
55	Contracts should implement a two-step ownership transfer	Informational	Acknowledged
56	Hardcoded packet version for cross-chain requests	Informational	Acknowledged
57	Dead code in `handle_deposit_info_update` function	Informational	Acknowledged
58	Missing emission of message hash attribute	Informational	Resolved
59	Emitted source chain ID is invalid	Informational	Resolved
60	Unnecessary state rollbacks implemented	Informational	Resolved
61	Unneeded payable annotations	Informational	Resolved
62	Inconsistent amount validation for `MAX_TRANSFER_SIZE`	Informational	Resolved
63	Inconsistent pause modifier for the `iReceive` function	Informational	Acknowledged
64	Usage of deprecated functionality	Informational	Resolved
65	Fees are not charged for `SetDappMetadata` and `ISend` messages	Informational	Resolved
66	Vault address is not validated	Informational	Resolved
67	Nonce instantiation in CosmWasm `gateway` contract differs from other implementations	Informational	Resolved
68	Unneeded custom reply identifiers for `ReplyOn::Never` submessages	Informational	Resolved
69	Storage costs in NEAR contracts are borne by the deployer	Informational	Acknowledged

Detailed Findings

1. Incorrect reentrancy unlock causes denial of service

Severity: Critical

In `asset-forwarder-contracts:near/contracts/asset-forwarder/src/execution.rs:205`, the `i_deposit_with_message` function calls `set_deposit_lock` to revert the `deposit_lock` when the caller deposits native funds. This is incorrect because the `deposit_lock` is used to prevent a reentrancy attack, as seen in line `125`. Consequently, `i_deposit` and `i_deposit_with_message` functions cannot be unlocked, causing a DOS of the `asset-forwarder` contract.

This issue also exists in line `245` where the `ideposit_info_update` function incorrectly calls the `set_deposit_lock` function to unlock the `deposit_info_lock` reentrancy lock.

Recommendation

We recommend modifying line `205` to use `set_deposit_with_message_lock` and line `245` to use `set_deposit_info_lock`.

Status: Resolved

2. Attackers can create deposit requests without including funds

Severity: Critical

In `asset-forwarder-contracts:near/contracts/asset-forwarder/src/execution.rs:145-158`, the fund deposit event `FundsDeposited` is logged in line `145` before performing the validations in lines `147` and `155`. If the caller does not attach sufficient funds or the caller is not the intended source token, the event will still be included in the transaction receipt with its status as a failure.

This is problematic because the orchestrator does not validate whether the receipt's status is a success or failure when listening to the events. The orchestrator will fetch and parse the event as long the receiver ID is the gateway address and the regex matches. This means that even if an error occurs during the fund deposit in the `AssetForewarder` NEAR contract, the orchestrator still processes the events to submit the transaction to the destination chain.

This allows an attacker to initiate a deposit request on the source chain that purposefully fails after the event is logged. Since the orchestrator incorrectly processes the failed event, the forwarder will bridge the funds to the destination chain and is subsequently able to claim the eligible funds on the source chain. As the fund deposit on the source chain never succeeded, the forwarder is either unable to receive the claimable funds or, if sufficient liquidity is available on the source chain, incorrectly receives funds that are intended for other forwarders. Consequently, a loss of funds scenario will occur.

This issue also exists in lines `198` and `253` for the `i_deposit_with_message` and `ideposit_info_update` functions.

Recommendation

We recommend modifying the event logging order so it is only logged after successful executions. For example, line `145` should be removed and placed after lines `151` and `160`.

Status: Resolved

3. Unauthorized `ibc_channel_connect` may lead to unauthorized calls

Severity: Critical

The CosmWasm `gateway` contracts are designed to be deployed on WASM-enabled appchains like Osmosis and Juno. The Router protocol uses IBC channels instead of orchestrators to relay messages through these `gateway` contracts. This is because IBC inherently ensures message integrity, eliminating the need to verify Router Chain validator set’s signature, as is required on other chains like NEAR or EVM.

To enable communication via IBC channels, WASM-enabled application chains are provided with an `ibc_channel_connect` and other hooks in smart contracts. To authenticate the source channel when using IBC, there are two methods: object capabilities and source authentication. Since application chains may not know which chain a smart contract will enable IBC with, it is the smart contract's responsibility to validate the source chain.

In the context of CosmWasm `gateway` contracts, there is currently no authentication for channels. This opens up a potential vulnerability where an attacker can create a malicious IBC appchain and send a connection request to the `gateway` contract.

Specifically, the `ibc_channel_connect` function in `cosmwasm-gateway-contract:src/contract.rs:290-303` will accept any request and store it in `CHANNEL_ID` for further packet sending. This malicious channel can overwrite the legitimate channel, causing the `gateway` contract to execute messages from the malicious chain instead of the intended Router Chain.

Recommendation

We recommend verifying the `channel` against some preset configurations.

Status: Resolved

4. Nonce is incorrectly incremented when fungible tokens transfer fails

Severity: Critical

In `asset-forwarder-contracts:near/contracts/asset-forwarder/src/execution.rs:555`, the `irelay_transfer_callback` function stores the previously, in line `521` incremented `deposit_nonce` nonce, when the `ft_transfer` promise results in a failure.

This is problematic because a failed fungible token transfer will halt the execution and not emit the `FundsPaidWithMessage` event. Hence, the nonce is unused and does not need to be incremented.

Consequently, the required sequential order of the nonce is not guaranteed, causing the orchestrators not to process future `iRelay` messages and ultimately causing a loss of funds for future token payers.

Recommendation

Instead of incrementing the `deposit_nonce` in line `521` and supplying the incremented nonce to the `handle_message_callback` function in line `548`, we recommend incrementing and storing the nonce in this callback function shortly before consuming the nonce.

Additionally, line `555` can be removed, so that the incremented but unused nonce is not stored in case the fungible token transfer fails.

Status: Resolved

5. Attackers can register and deregister their accounts for profit

Severity: Critical

In `asset-forwarder-contracts:near/contracts/token/src/lib.rs:111`, the `register` function will register 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.

Recommendation

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

Status: Resolved

6. Lowercasing case-sensitive addresses causes unexpected behavior and loss of funds

Severity: Critical

The CosmWasm middleware contract keeps track of various balances for addresses, such as depositors and forwarders. The addresses may have different formats depending on the chains. For example, TRON (base58 address format); Solana, and Polkadot use case-sensitive addresses; whereas the Ethereum, NEAR, and Cosmos chains use case-insensitive addresses.

The issue occurs where the CosmWasm middleware contract lowercase addresses in many places. For instance, when calling the WithdrawBlockedFunds message, the supplied depositor address is lowercased in asset-forwarder-middleware:contracts/asset-forwarder/src/execution.rs:70. Subsequently, this lowercase address is used as the recipient for the token transfer in line 174, resulting in incorrect and potentially failing token transfers during the iReceive function call. For example, lowercasing the base58 address format for a TRON account results in an invalid address.

Consequently, interacting with the CosmWasm middleware contract from a chain with case-sensitive addresses results in unexpected behavior and, in the worst case, lost and unrecoverable funds.

Recommendation

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

Status: Resolved

7. Incorrect tokens are used to account for claimable tokens, causing a loss of funds

Severity: Critical

The `update_chain_liquidty_and_claimable_amount` function in `asset-forwarder-middleware:contracts/asset-forwarder/src/execution.rs:807-888` updates the liquidity and claimable amounts as part of the fund deposit.

Additional fees, possibly in different tokens, can be supplied to an existing fund deposit on the source chain to incentivize forwarders further. Those extra fees are added to the forwarder's claimable funds by calling the `update_claimable_amount` function in lines `867-873`, making the forwarder eligible to claim those extra fees on the source chain.

However, the supplied `token` is incorrectly set to the source token (`request_info.src_token`), the token that has been deposited on the source chain, instead of the extra fee token. As a result, the extra fees are withdrawn from the wrong token balance.

This can be exploited by adding a low-value extra fee token (e.g., USDC, equalling 1 USD) and, in return, being able to claim the same amount of the higher value `src_token` (e.g., ETH) on the source chain and thus stealing funds.

Similarly, the system fees are also accounted for with the wrong token in line `879`.

Recommendation

We recommend modifying lines `870` and `879` to use the extra fee token (`extra_fee.token`) instead of the source token.

Status: Resolved

8. Attackers can steal funds by repeatedly withdrawing blocked funds

Severity: Critical

Due to not deducting the withdrawn blocked amount in the `BLOCKED_FUNDS` storage in `asset-forwarder-middleware:contracts/asset-forwarder/src/execution.rs:134`, a depositor can withdraw their blocked funds multiple times because the `BLOCKED_FUNDS` storage state is not mutated.

Consequently, attackers can steal funds from other depositors, causing a loss of funds scenario.

Recommendation

We recommend updating the `BLOCKED_FUNDS` state by deducting the withdrawn amount.

Status: Resolved

9. Erroneous claimable updates lead to the loss of the forwarder's funds

Severity: Critical

In `asset-forwarder-middleware:contracts/asset-forwarder/src/execution.rs:834`, the `update_chain_liquidty_and_claimable_amount` function updates the forwarder's claimable liquidity without adding the previous liquidity amount. Specifically, the `update_claimable_amount` will overwrite the `CLAIMABLE_AMOUNT` storage state in line `1276`. Consequently, a forwarders' existing claimable amount will be overwritten, causing a loss of funds.

Similarly, in the following instances, the existing `CLAIMABLE_AMOUNT` is not included:

- `asset-forwarder-middleware:contracts/asset-forwarder/src/execution.rs:843`
- `asset-forwarder-middleware:contracts/asset-forwarder/src/execution.rs:872`
- `asset-forwarder-middleware:contracts/asset-forwarder/src/execution.rs:881`

Recommendation

We recommend retrieving the existing liquidity using the `fetch_claimable_amount` function and increasing it before calling the `update_claimable_amount` function.

Status: Resolved

10. Extra fees can be unboundedly added to a deposit, resulting in failure to conclude the deposit

Severity: Critical

In `asset-forwarder-middleware:contracts/asset-forwarder/src/sudo.rs:125-226`, the `handle_deposit_info_update` function handles the `SudoMsg::HandleDepositInfoUpdate` sudo message, effectively either initiating a refund request or adding extra fees to the existing deposit.

Anyone on a source chain can initiate this deposit update, not just the depositor. Consequently, as there is no maximum limit to the number of extra fees that can be added to a deposit, an attacker can add a high number of extra fees to a deposit, resulting in unbounded loop iterations whenever the fees that are stored in the deposit's RequestInfo.extra_fee are iterated.

Specifically, this occurs in asset-forwarder-middleware:contracts/asset-forwarder/src/execution.rs:851-885 and in lines 265-270 of the CosmWasm middleware contract.

Additionally, those extra fees are also iterated and transferred to either the depositor or forwarder within the iReceive function of the Solidity and NEAR asset-forwarder contracts. In the case of the Solidity asset-forwarder contract, this will most likely result in an out-of-gas error, reverting the transaction. As a result, the funds will be stuck.

Recommendation

We recommend setting a reasonable maximum limit for extra fees that can be added to a deposit.

Status: Acknowledged

The client states that they will add logic to update the extra fee object if the same token is added again as an extra fee. In that way, they will have a max length equal to the whitelisted token count. In addition, the client states that the maximum length will be limited to 10 in future.

11. Updating the forwarder's claimable amount during a withdrawal request uses the wrong decimal precision, resulting in the ability to steal funds

Severity: Critical

The create_withdraw_request function in asset-forwarder-middleware:contracts/asset-forwarder/src/execution.rs:565-728 processes a withdrawal request for the forwarder and updates the forwarder's claimable amount by calling the update_claimable_amount function in lines 646-652 to deduct the withdrawn amount.

The forwarder's claimable amount is stored in the CLAIMABLE_AMOUNT state, which is normalized to 18 decimals to account for different token decimals on different chains.

Specifically, in lines 1273 to 1276, the update_claimable_amount function fetches the token’s decimals and scales the given amount to 18 decimal places (DEFAULT_DECIMAL). This means that it expects the supplied amount in the decimal precision of the token on the given chain (e.g., if the token is USDC on ETH Mainnet, the amount should have 6 decimals).

However, the `create_withdraw_request` function in line `628` passes the `withdraw_info[i].amount` decimal values as 18 decimals, which is incorrect because it is not the token's decimals. This would cause the `update_claimable_amount` function to normalize the amount incorrectly.

For instance, if a forwarder attempts to withdraw an amount of `500e18`, the `dest_token` token uses 6 decimals, and the remaining claimable amount should be `500e18`. However, the `update_claimable_amount` function will incorrectly calculate it and store a massively inflated value due to `500e18 * 10^(18 - 6) = 500e30`.

Consequently, the computed claimable amount will be incorrect, allowing the forwarder to withdraw more funds than expected, causing a loss of funds scenario.

The same issue is observed in the `handle_sudo_ack` function in `asset-forwarder-middleware:contracts/asset-forwarder/src/sudo.rs:259-265`, handling the error case from a received acknowledgment message and reverting the claimable amount state change to be able to repeat the withdrawal request.

Recommendation

We recommend normalizing the `claimable_amount` to match the token's decimal precision and deducting the sanitized withdraw amount in line `651` to ensure the `update_claimable_amount` function receives the correct amount.

Status: Resolved

12. Incorrect access control and entry point implementation

Severity: Critical

In several entry points of the CosmWasm `gateway` contract, incorrect access control is implemented.

Firstly, the `SetVault` message in `cosmwasm-gateway-contract:src/contract.rs:573` allows anyone to update the vault address, which is responsible for minting and burning `ROUTE` tokens. This message should be validated to ensure only the contract owner can call this to prevent failure when interacting with `ROUTE` tokens.

Secondly, the `ExecuteSelfForIAck` message in line `548` should ensure the caller is the contract itself because it will only be executed in line `475` by the `iack` function.

Lastly, mock messages that are for testing purposes, such as `SetNonce`, `MockIReceive`, `Init`, and `MockIAck` (see lines `522`, `603`, `576`, `694`) should be removed as they represent a source for vulnerabilities to occur. For example, an attacker can call the `MockIReceive` message with any arbitrary `route_amount` to mint infinite `ROUTE` tokens.

Recommendation

We recommend applying the recommendations mentioned above.

Status: Resolved

13. Risk of inaccurate account ID determination

Severity: Critical

The `ft_on_transfer` function in `asset-forwarder-contracts:near/contracts/asset-forwarder/src/execution.rs:34-112` of the NEAR `asset-forwarder` contract acts as the entry point to the deposit and relay functionality. Specifically, this function is either called directly by the user if native NEAR tokens are involved or indirectly by a NEP-141 fungible token contract via the `ft_transfer_call` function.

Due to this two-fold logic, determining the specific depositor and forwarder account ID is not trivial. Using the `env::signer_account_id` function to figure out the depositor account ID is potentially incorrect, as it is the signer of the transaction, ignoring any intermediary contract. Further, the `env::predecessor_account_id` function, returning the caller's account ID, is also not necessarily suitable, as the caller might be a fungible token contract.

As a result, in lines `142`, `194`, `238`, `359`, `485`, and `506`, the depositor or forwarder account IDs are potentially incorrect, resulting in unexpected behavior.

Concretely, in line `238`, the `ideposit_info_update` function determines the depositor by calling the `env::predecessor_account_id` function, which is the contract caller. This is incorrect in the case of a fungible token contract, as this would set the depositor to the token contract's account ID.

Consequently, excess fee funds will be incorrectly sent to the fungible token contract instead of the actual depositor, causing a loss of funds.

Recommendation

We recommend propagating the account ID in the provided `sender_id` parameter of the `ft_on_transfer` function, supplied by the calling fungible token contract, or, in the case of a native NEAR token transfer, using the `env::predecessor_account_id` function.

Extra care should be taken when using the `sender_id` parameter, as the caller can arbitrarily choose if native NEAR tokens are involved.

Status: Resolved

14. Continuously failing ERC-20 token transfers in the `iReceive` function leads to stuck funds

Severity: Critical

The `iReceive` function in `asset-forwarder-contracts:evm/src/AssetForwarder.sol:384-406` handles received cross-chain requests originating from the Router's CosmWasm `middleware` contract. Specifically, a specified list of ERC-20 tokens is transferred to the respective recipients as part of withdrawing blocked funds, as well as the refund mechanism, and a forwarder's withdrawal request.

However, if a single ERC-20 token transfer fails, the `iReceive` function will revert, and the cross-chain request will be marked as failed.

If the token transfer continues to fail (e.g., due to the token contract being paused or the recipient being blocked), retrying the failed request via the `middleware` contract will still be unsuccessful, causing other tokens to be stuck and preventing them from being sent to the recipient.

Recommendation

We recommend utilizing a mechanism similar to that observed in the NEAR `asset-forwarder` contract, where the failed token transfers are accounted for, and the recipient can claim those individual tokens later on, as seen in `near/contracts/asset-forwarder/src/execution.rs:744-756`.

Status: Resolved

15. Timeout for `SendPacket::ReceivePayload` IBC packets are not handled, resulting in lost `ROUTE` tokens

Severity: Critical

The CosmWasm `gateway` contract receives `iReceive` and `iAck` and sends `iSend` messages from and to Router Chain via IBC. In `cosmwasm-gateway-contract:src/contract.rs:505-512`, the `ibc_packet_timeout` function is called when a timeout occurs for an IBC packet.

However, in the current implementation of the `gateway` contract, this function does not handle the timeout for such packets. Consequently, already burned `ROUTE` tokens, as part of the `iSend` call, are not refunded and remain burned as the cross-chain message can not be relayed.

Recommendation

We recommend adding special handling for `iSend` message timeouts, specifically `SendPacket::ReceivePayload` IBC packets.

Status: Resolved

16. `HandlerExecMsg::IReceive` message is sent to the handler contract regardless of the `ASM` contract execution result

Severity: Critical

As part of processing `iReceive` messages, an optionally specified Additional Security Modules (ASM) contract is called in `cosmwasm-gateway-contract:src/contract.rs:683-701`, determining whether the message is permitted to be relayed to the handler contract. However, the result of the `ASMMsg::Verify` message is not considered, and the `HandlerExecMsg::IReceive` message is sent to the handler contract regardless. Consequently, the `ASM` contract validation is bypassed, resulting in the execution of unauthorized `iReceive` messages.

Recommendation

We recommend creating a separate submessage for the `ASM` contract call and, depending on the result, continuing with the `ROUTE` token minting and sending the `HandlerExecMsg::IReceive` message to the handler contract.

Status: Resolved

17. Unordered IBC channel is incorrectly enforced, resulting in out-of-order IBC packets

Severity: Critical

In `cosmwasm-gateway-contract:src/contract.rs:261-287`, the `ibc_channel_open` function is called upon to open an IBC channel to validate the message ordering and the counterparty version. The channel message order is expected to be `IbcOrder::Ordered`, which is checked in line `268`. However, the check is inverted and only allows `IbcOrder::Unordered` channels. Consequently, only unordered IBC channels are supported, resulting in IBC packets being relayed out of order.

Recommendation

We recommend inverting the logic in line `268` only to allow `IbcOrder::Ordered` channels.

Status: Resolved

18. Incorrect data parsing for HandlerExecMsg::IReceive and HandlerExecMsg::IAck messages resulting in lost ROUTE tokens

Severity: Critical

The CosmWasm gateway contract receives iReceive messages from Router Chain via IBC and processes them in the ireceive function in cosmwasm-gateway-contract:src/contract.rs:392-451.

Specifically, the ireceive function dispatches the ExecuteMsg::SelfExecForIReceive submessage to itself, which is handled by the self_exec_for_ireceive function in lines 628-760 and takes care of validation, ASM contract execution, minting ROUTE tokens, and sending the HandlerExecMsg::IReceive message to the specified handler contract.

Subsequently, the handle_self_exec_for_ireceive_reply function handles the reply for the previously sent ExecuteMsg::SelfExecForIReceive message and responds with the AckPacket::AckReceivePayload to Router Chain, concluding the iReceive message processing.

However, if the HandlerExecMsg::IReceive message call fails, lacks a response, or returns empty data, set via the Response::set_data function, attempting to parse the reply in the handle_ireceive_reply function in lines 88-93 errors and thus reverting the state of the ExecuteMsg::SelfExecForIReceive submessage, including minting ROUTE tokens, the execution status in SRC_EXECUTED, and the ASM contract call. Nevertheless, the AckPacket::AckReceivePayload is sent to Router Chain, indicating a failed but processed iReceive message.

Consequently, this can lead to lost ROUTE tokens that have been burned on the source chain but not minted on the destination chain due to the token minting being reverted.

Similarly, the same issue is observed for processing iAck messages, where a failed HandlerExecMsg::IAck message contract call results in reverting the state and responding with AckPacket::AckReceiptPayload to Router Chain.

Recommendation

We recommend parsing the HandlerExecMsg::IReceive and HandlerExecMsg::IAck message without returning an error from the reply function, thus not reverting the state of the ExecuteMsg::SelfExecForIReceive and ExecuteMsg::SelfExecForIAck submessages.

Status: Resolved

19. Panic in handler callback causes denial of service and loss of funds

Severity: Critical

In asset-forwarder-contracts:near/contracts/asset-forwarder/src/execution.rs:602, the handle_message_callback function panics if the execution data cannot be parsed. In this case, the set_relay_with_message_lock function in line 629 will not be executed, causing the i_relay_with_message function to fail in line 428 because the relay_with_message_lock is still locked.

Additionally, the user funds sent in lines 476 and 492 are not refunded and will be stuck in the contract, causing a loss of funds for the user.

Recommendation

We recommend modifying the implementation so the exec_flag is set to false with an empty exec_data if the execution data cannot be parsed correctly.

Status: Resolved

20. Inability to withdraw pending forwarder funds if there is no matching fund deposit

Severity: Major

The handle_funds_paid function in asset-forwarder-middleware:contracts/asset-forwarder/src/execution.rs:730-805 handles the SudoMsg::HandleFundsPaid sudo message and is called once a forwarder has relayed and paid the funds to the depositor on the desired destination chain.

This function properly handles the case when a forwarder prematurely pays funds without a previously registered deposit and temporarily accounts for those funds in the PENDING_FUND_PAID storage map. A future corresponding deposit will then be matched with the pending funds, and the forwarder can withdraw the funds on the source chain.

Suppose a forwarder prematurely pays funds, and the corresponding deposit is never registered. In that case, the forwarder's pending funds can not be withdrawn and remain locked indefinitely, as indicated in the comment in line 782.

Recommendation

We recommend implementing a withdrawal mechanism so forwarders can withdraw pending funds paid after a period of time.

Status: Acknowledged

21. Failed `ASM` contract call or `ROUTE` token minting does not abort the whole transaction, causing partial state to be committed

Severity: Major

Received `iReceive` messages via IBC are handled by the `ireceive` function in `cosmwasm-gateway-contract:src/contract.rs:415`, processed within the `ExecuteMsg::SelfExecForIReceive` submessage configured as `ReplyOn::Always` that does not revert the entire transaction if an error occurs (see lines `195-199`).

This means that if the submessage call errors when performing external calls, the entire transaction state is not reverted, causing a partial state committed with a failed request acknowledged back to Router Chain. This is problematic because Router Chain would interpret the request to fail entirely without expecting some of the sub-requests to succeed.

For example, in lines `683-759`, the order of the messages is ASM contract validation, mint `ROUTE` tokens, and handler address execution. If the first two messages succeed but the last one fails, it would cause the ASM contract state and minted `ROUTE` tokens to persist.

Similarly, replay validation errors while processing `iAck` messages in lines `776-795` also do not abort the whole transaction and lead to the `AckPacket::AckReceiptPayload` acknowledgment being sent to Router Chain with the `exec_status` set to `false`.

Recommendation

We recommend reverting the sub-message state if the request fails. Additionally, we recommend conducting the validations for `iReceive` and `iAck` requests in the `ireceive` and `iack` functions to ensure the whole transaction aborts in case of an error.

Status: Acknowledged

22. The reentrancy lock mechanism in the NEAR `asset-forwarder` contract can be abused to grief the contract

Severity: Major

Nearly all functions in the NEAR `asset-forwarder` 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, 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.

For instance, the i_relay_with_message function in asset-forwarder-contracts:near/contracts/asset-forwarder/src/execution.rs:423-511 ensures that the relay_with_message_lock mutex is set to false at the beginning of the function and reverts otherwise. Thereafter, the function is locked by calling the set_relay_with_message_lock function in line 432. As long as all the callbacks are not executed, repeatedly calling this function will panic. Once the last callback function handle_message_callback is executed, the lock is released in line 629.

However, this lock mechanism opens up a potential denial-of-service (DoS) attack 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 from all functions and ensuring that the contract's state is not exploitable between callbacks. Specifically, ensure event nonces are only incremented in the same calling context as the event is emitted.

Status: Resolved

23. Admin cannot withdraw native tokens from the asset-forwarder contract

Severity: Major

In asset-forwarder-contracts:evm/src/AssetForwarder.sol:154, the iDepositUSDC function ensures that the caller sent native tokens equal to the fee amount. However, there is no entry point for the admin to withdraw such tokens from the contract. Consequently, the funds are stuck in the contract.

Recommendation

We recommend implementing a withdrawal function for the admin to withdraw native tokens.

Status: Resolved

24. Hardcoded gas limits might cause cross-chain messages to fail

Severity: Major

In `asset-forwarder-middleware:contracts/asset-forwarder/src/execution.rs:187`, hardcoding the `dest_gas_limit` to `500_000` might result in transaction failure on destination chains due to an out-of-gas issue. This is because different chains have different gas limits, and the hardcoded gas might be too little or too big for those chains.

Recommendation

We recommend simulating transactions and using setter methods to adjust gas limits, allowing for future updates dynamically.

Status: Resolved

25. Depositors are unable to withdraw blocked funds due to incorrect address format

Severity: Major

Depositors can withdraw blocked funds by sending the `ExecuteMsg::WithdrawBlockedFunds` message to the CosmWasm `middleware` contract, which will subsequently send a cross-chain request to the specified source chain to transfer the blocked funds to the depositor.

To compensate for the cross-chain request fees, the depositor has to attach a certain amount of `ROUTE` token funds to the message. If the paid fees are higher than the actual fees, the leftover `ROUTE` tokens are refunded to the depositor via a native bank message in `asset-forwarder-middleware:contracts/asset-forwarder/src/reply.rs:157-163`.

However, the provided recipient address in line `158` is the depositor's address on the source chain, for instance, an Ethereum address, instead of the expected Router Chain Cosmos address. As a result, the transfer fails, and the whole transaction reverts, preventing the blocked funds from being withdrawn.

Recommendation

We recommend using the caller of the `ExecuteMsg::WithdrawBlockedFunds` message (i.e., `info.sender`) as the recipient of the native `ROUTE` fee refund.

Status: Resolved

26. Extra fees added to a relayed deposit cannot be withdrawn

Severity: Major

A depositor can add extra fees to a deposit to further incentivize forwarders to ensure a timely relay of the funds to the destination chain. Such a deposit update can be initiated on the source chain, for instance, by calling the `iDepositInfoUpdate` function on the Solidity `asset-forwarder` contract, which will then transfer and escrow the specified fees.

Subsequently, the CosmWasm `middleware` contract receives the `SudoMsg::HandleDepositInfoUpdate` sudo message and handles the deposit update in the `handle_deposit_info_update` function in `asset-forwarder-middleware:contracts/asset-forwarder/src/sudo.rs:125-226`. In case there is no matching deposit, or the fee token is not whitelisted, the fee amount is temporarily accounted for in the `BLOCKED_FUNDS` storage map and can be withdrawn later on.

However, if the deposit is already paid, i.e., `is_paid` is set to a value other than `None`, the added fees are regularly added to the deposit. As the deposit is already paid and the forwarder's claimable funds are already updated, the newly added fees are locked and can not be withdrawn.

Recommendation

We recommend checking in the `handle_deposit_info_update` function to see if the deposit has already been paid, and if so, add the fees to `BLOCKED_FUNDS`.

Status: Acknowledged

27. User's `create_refund_request` can grief forwarders, preventing fund retrieval

Severity: Major

When a user deposits funds in the source chain for bridging to the destination chain and wishes to reclaim those funds on the source chain (for reasons such as the request not being fulfilled on the destination chain), they can create a refund request in the `middleware` contract.

In `asset-forwarder-middleware:contracts/asset-forwarder/src/execution.rs:781-785`, the middleware performs a crucial check to ensure that the user has not already been paid on the destination chain. This verification is achieved by checking the status of `request_info.is_paid`, ensuring it is either `None` or not.

This check prevents refund requests for requests that have already been paid. However, there is the following edge case to consider: Users can create a refund request immediately after the forwarder has paid them on the destination chain. Since the event from the destination chain must go through Router Chain orchestrator consensus to confirm that the forwarder has indeed made the payment, users can exploit this delay to receive payment on the destination chain and simultaneously request a refund on the source chain, resulting in a griefing situation for the forwarder and double spending for the user.

Certain checks are in place to prevent these scenarios. Users are not allowed to create a refund request before a specified expiry parameter:

```
env.block.time.seconds() > request_info.expiry_timestamp
```

Additionally, it is ensured that the destination chain is synchronized with the middleware:

```
request_info.expiry_timestamp                                    <
fetch_last_info_received(deps.as_ref(),
&request_info.dest_chain_id)
```

The `fetch_last_info_received` function updates `last_info_received` when the middleware processes events from the destination chain.

However, it is important to note that even with these checks, certain cases can bypass them. For example, an attacker can initially create a request with low fees, which may not incentivize forwarders to pick it up. As the request is about to expire (approaching `request_info.expiry_timestamp`), the user can increase the fee in the `deposit_info_update` (which does not change the expiry). This would prompt forwarders to pick up the request, and immediately after it is fulfilled on the destination chain, the attacker can create a refund request, exploiting the system.

Recommendation

We recommend removing the `CreateRefundRequest` message. Users should only be able to initiate withdrawal requests on the source chain. This would allow forwarders to prove that they already deposited funds on the destination chain.

Status: Resolved

28. The sender chain will not be notified of errors via IBC acknowledgment

Severity: Major

In `cosmwasm-gateway-contract:src/contract.rs:333`, the `ibc_packet_receive` function returns a `Result` that consists of either `IbcReceiveResponse` or `StdError`. This is problematic because if an error occurs, it will prevent the packet from being acknowledged back to the sender chain.

Consequently, the sender chain will not receive any response, causing the sender contract's `ibc_packet_timeout` entry point to be entered instead of `ibc_packet_ack` entry point (where it expects an acknowledgment). This might cause unexpected outcomes if the sender contract behaves differently for both entry points.

Recommendation

We recommend encoding the error and returning it as an acknowledgment.

Status: Resolved

29. Handling `RequestPayload` IBC packet will fail due to insufficient integer values

Severity: Major

In `cosmwasm-gateway-contract:src/contract.rs:416`, the `ireceive` function attempts to cast the `route_amount` into a `u64` integer variable. This is problematic because when the `RequestPayload` packet is sent from Router Chain, the `Int` variable is used, which can hold up to a `u256` value. This means there is a possibility that the transacted `ROUTE` tokens are larger than the `u64` value, which will cause an error in the `string_to_u64` function in `cosmwasm-gateway-contract:src/helper.rs:4`.

Additionally, the `ROUTE` token in the Ethereum mainnet uses 18 decimal places. When calling the `iSend` functions in the Solidity contract, the `ROUTE` amount is specified as the `uint256` parameter, indicating there is a high chance that the `ROUTE` tokens will be transacted to a value larger than `u64`.

Recommendation

We recommend modifying the `route_amount` to use the `u256` variable type.

Status: Resolved

30. Incorrect system fee calculation when computing fee distribution

Severity: Major

In `asset-forwarder-middleware:contracts/asset-forwarder/src/queries.rs:481`, the `calculation_fee_distribution` function computes the system fee by multiplying the fee amount with the `ROUTER_VALIDATION_FEE` and dividing it by the token price in USD. This is problematic because the fee is not charged as a percentage after multiplying with the `ROUTER_VALIDATION_FEE`. As there is no denominator to divide the multiplied amount, the fee amount will be larger than intended.

Other than that, the system fee computation charges the fee in USD denom by dividing the token amount by the token price. This is incorrect because multiplication should be used to compute the queried token's price correctly.

Consequently, the system fee computed will be incorrect, causing users to be charged with incorrect fees.

Recommendation

We recommend computing the price by multiplying the token amount by the token price and adding a denominator to charge the fee as a percentage correctly. To avoid rounding errors when computing the fee, a 10000 bps value denominator can be implemented.

Status: Resolved

31. Lack of entry point to remove outdated storage entries

Severity: Major

In `asset-forwarder-middleware:contracts/asset-forwarder/src/execution.rs:987-991`, the `set_whitelist_contracts` function updates the `WHITELISTED_ADDRESSES` with the (chain ID, voyager contract address) key entry to `true`. As there are no entry points for the owner to remove the old storage, updating the chain to use a new voyager contract address will still reflect the old contract address as valid.

Suppose the owner whitelists the first contract for the Ethereum mainnet chain ID. If the owner decides to use a second contract and not trust the first contract anymore, the `WHITELISTED_ADDRESSES` storage will still reflect that the first contract for the Ethereum mainnet chain ID is still whitelisted, which is incorrect.

This issue also affects the `TOKEN_TO_SYMBOL_MAPPING` storage in lines `940` to `947` and the `CHAIN_BYTES_INFO` storage in line `1144`. The former will be incorrect when updating the (new source chain ID, symbol) key entry to a new source token because the old source token with (old source chain ID, symbol) key entry will persist in storage. The latter will be incorrect when updating the (new chain ID) key entry because the (old chain ID) key entry will persist in storage.

Recommendation

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

Status: Acknowledged

32. Denial of service due to unbounded processing of forwarder balances

Severity: Major

In `asset-forwarder-contracts:near/contracts/asset-forwarder/src/execution.rs:666-707`, the `i_receive` function calls the `get_all_forwarder_balance` function to retrieve all pending forwarder balances and dispatches token transfer promises. If any error occurs when sending the tokens, the `i_receive_callback` function will add the funds back to the forwarder balance, as seen in lines `750` to `755`.

This is problematic because all the pending funds are processed unboundedly, causing the transaction to fail due to an out-of-gas error. Consequently, this affects both the `i_receive` and `forwarder_withdraw_refund` functions as they process all forwarder balances unboundedly.

To trigger such an attack, an attacker needs to add tokens to the victim's balance through the `middleware` contract that will fail on purpose, so it increases the storage entry. An example of this is creating fake fungible tokens that will fail when the `ft_transfer` function is called. To add the failed fungible token to the victim's balance, the attacker calls the `ideposit_info_update` function with the victim's `deposit_id` in line `219` to emit a `DepositInfoUpdateEvent` event.

The event will be handled by the `HandleDepositInfoUpdate` sudo message in `asset-forwarder-middleware:contracts/asset-forwarder/src/sudo.rs:84`. Since the attacker's token is not whitelisted, lines `186` to `199` will be entered, causing the fake fee token to be stored in the `BLOCKED_FUNDS` storage as (source chain ID, victim, fake token) key entry.

Afterward, the attacker calls the `WithdrawBlockedFunds` message in `asset-forwarder-middleware:contracts/asset-forwarder/src/execution.rs:59` to initiate a withdrawal for the victim. Using the key entry in the previous paragraph as the `chain_id`, `depositor`, and `token` parameter, the attacker essentially calls the `middleware` contract to initiate a cross-chain call to refund the fake token to the victim, as seen in lines `134` to `214`.

The refund will ultimately call the `i_receive` function in `asset-forwarder-contracts:near/contracts/asset-forwarder/src/execution.rs:633`. Since the attacker's fake token will fail when `ft_transfer` is called, the `i_receive_callback` function will add the fake token entry to the victim's `forwarder_balances`, causing a denial of service attack to the `i_receive` and `forwarder_withdraw_refund` functions.

We classify this issue as major due to the complexity and difficulty of the attack. For instance, the attacker must pay `ROUTE` tokens to initiate the cross-chain call, as seen in `asset-forwarder-middleware:contracts/asset-forwarder/src/reply.rs:149-153`.

Recommendation

We recommend implementing a pagination mechanism in the `forwarder_withdraw_refund` function so users can choose which tokens to withdraw. This prevents the out-of-gas error because withdrawals are performed in small batches. Additionally, consider modifying the `i_receive` function to only process the sent funds instead of all the pending forwarder funds.

Status: Resolved

33. Refund request fees are borne by the `middleware` contract

Severity: Major

In `asset-forwarder-middleware:contracts/asset-forwarder/src/execution.rs:228`, the `create_refund_request` function does not charge any outbound fees from the caller. As a comparison, claims and refunding blocked funds requests are borne by the caller, as seen in `asset-forwarder-middleware:contracts/asset-forwarder/src/reply.rs:46` and line `149`.

Consequently, the `middleware` contract will incur the cost of refund request fees, causing a loss of funds scenario.

Recommendation

We recommend charging the outbound fees from the caller when initiating a refund request.

Status: Acknowledged

34. Extra fee token's liquidity is updated with an incorrect value

Severity: Major

In `asset-forwarder-middleware:contracts/asset-forwarder/src/execution.rs:863`, the `update_chain_liquidity` function updates the fee token's liquidity with the `request_info.src_amount` amount. This is incorrect because the liquidity increase should be the total amount of extra fee tokens sent, not the deposited amount of funds on the source chain.

Consequently, the fee token's liquidity will increase incorrectly, reflecting an incorrect state and potentially causing an underflow error in line `658`.

Recommendation

We recommend modifying line 863 to increase the fee token’s liquidity by extra_fee.sys_fee, extra_fee.forwarder_fee and extra_fee.partner_fee.

Status: Resolved

35. Failure to decode the recipient address leads to loss of funds

Severity: Major

In asset-forwarder-middleware:contracts/asset-forwarder/src/execution.rs:495, the convert_address_from_bytes_to_string function is called to decode the recipient address based on the destination chain type. If an error occurs, the unwrap function will be triggered, causing the transaction to panic and abort.

This is problematic because the execution flow expects any errors inside the fund_deposit_post_processing function to be handled gracefully. Ideally, if an error occurs, the sent funds should be stored in the BLOCKED_FUNDS storage state for the depositor to withdraw later, as seen in lines 365 to 380.

Consequently, errors when decoding the recipient address will cause the sent funds to be stuck in the contract, leading to a loss of funds scenario.

Recommendation

We recommend modifying line 495 to use convert_address_from_bytes_to_string(&request_info.recipient, dest_chain_type)?; so the error can be propagated and handled in line 365.

Status: Acknowledged

36. Scaling decimal tokens might cause precision issues

Severity: Major

In asset-forwarder-middleware:contracts/asset-forwarder/src/queries.rs:503, the sanitize_amount function does not validate whether the resulting amount does not equal 0. This might happen when the user-provided amount is too small during division in line 506 due to integer rounding or an overflow occurs in line 513 when multiplying the decimal exponentials.

Consequently, the scaled decimal token amount will be incorrect, causing an incorrect amount of funds to be transacted.

Recommendation

We recommend modifying the sanitize_amount function to implement Result<T, E> and returning an error if the result is zero. If an error occurs, the funds can be added into the BLOCKED_FUNDS storage for the depositor to withdraw later.

Status: Acknowledged

37. Callback functions may run out of gas, resulting in inconsistent states of the NEAR asset-forwarder contract

Severity: Major

In several instances of the NEAR asset-forwarder codebase in asset-forwarder-contracts:near/contracts/asset-forwarder/src/execution.rs, 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.

For example, the irelay_transfer_callback callback function is scheduled for execution in lines 481-488 and 502-509 to handle the token transfer promise result. If the token transfer succeeds, lines 531 to 551 will be executed to call the handle_message function on the recipient address and the handle_message_callback internal function with 5 * TGAS static gas allocated each. This means that the irelay_transfer_callback function will need to consume at least 10 * TGAS of prepaid gas along with the necessary gas costs, or else the transaction will fail due to an out-of-gas error. Conversely, the tokens will be refunded to the caller if the transfer fails, as seen in lines 552 to 569.

The issue occurs when the i_relay_with_message function does not attach sufficient static gas to the callbacks due to insufficient gas initially provided by the caller, allowing the irelay_transfer_callback function to be executed with less than 10 * TGAS prepaid gas. The transaction will fail in this case because the NEAR compiler will try to attach 10 * TGAS to the handle_message and handle_message_callback functions, which is not possible. After all, the available prepaid gas is insufficient.

Consequently, the tokens sent by the caller will be stuck in the contract and not refunded, causing a loss of funds scenario.

This issue affects all callbacks with no static gas attached:

- i_relay_callback in lines 331 and 353.
- irelay_transfer_callback in lines 482 and 503.
- i_receive_callback in lines 705 and 815.

Recommendation

We recommend attaching a sufficient amount of static gas to all callback functions. 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.

In the above example, the `irelay_transfer_callback` function should be attached with at least `10 * TGAS`. Applying the `with_static_gas` function allows NEAR runtime to ensure there is sufficient prepaid gas for all promises and callbacks. If not, the transaction will abort without any state transitions.

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

38. Failed `iAck` messages can be replayed

Severity: Major

In `cosmwasm-gateway-contract:src/contract.rs:188-193`, the `handle_selfexec_for_iack_reply` function stores the `ACK_ACCEPTED1` and `ACK_ACCEPTED2` states as `true` to prevent replaying the packet. However, if the `iAck` request fails, the state is not updated.

This is problematic because requests should only be executed once despite their result status to prevent Router Chain from processing it multiple times.

Additionally, the current implementation differs from the Solidity and NEAR `gateway` contract implementations.

Recommendation

We recommend updating `ACK_ACCEPTED1` and `ACK_ACCEPTED2` to `true` even if the execution fails.

Status: Resolved

39. Chain pause mechanism is not enforced

Severity: Major

In asset-forwarder-middleware:contracts/asset-forwarder/src/execution.rs:1101 and line 1113, the pause_chain and unpause_chain functions allow the owner to pause and unpause chains. However, the pause mechanism is not consistently enforced throughout the codebase.

For example, when the Ethereum chain is paused, the handle_sudo_requests function in asset-forwarder-middleware:contracts/asset-forwarder/src/sudo.rs:21 still processes Ethereum cross-chain requests without restriction.

Recommendation

We recommend only processing cross-chain requests for unpaused chains.

Status: Resolved

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

Severity: Minor

In asset-forwarder-contracts:near/contracts/asset-forwarder/src/execution.rs:845 and asset-forwarder-contracts:near/contracts/token/src/lib.rs:129, 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.

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

41. Specifying a large dest_amount value could lead to funds overspending

Severity: Minor

To facilitate the bridging of assets from a source chain to a destination chain, users initiate the process by depositing funds on the source chain. These transactions trigger the emission of a FundsDeposited event in the contracts found at asset-forwarder-contracts:evm/src/AssetForwarder.sol:205 and asset-forwarder-contracts:near/contracts/asset-forwarder/src/execution.rs:138. This event carries essential details, including the source amount and destAmount.

Forwarders play a pivotal role by diligently monitoring these events and ensuring that the specified destAmount is securely transferred to the respective depositor on the destination chain.

Since no validation ensures the destAmount is not larger than the initial amount, malicious depositors can specify a higher destAmount than what they've actually locked, effectively allowing them to steal funds from the forwarders if the discrepancy between the source amount and the destination destAmount is not noticed.

We classify this issue as minor because specifying a destination amount larger than the sent amount will disincentivize the forwarders to relay funds.

Recommendation

We recommend modifying the implementation to remove destAmount from the event. When computing the fee distribution in the middleware contract, the fee charges can be applied to the amount and deducted directly.

Status: Acknowledged

42. CosmWasm gateway contract can not be paused in case of an emergency

Severity: Minor

Contrary to the Solidity and NEAR gateway contracts, the CosmWasm gateway contract implemented in cosmwasm-gateway-contract:src/contract.rs does not implement a pause mechanism to temporarily halt critical functionality, such as sending cross-chain requests via the ExecuteMsg::ISend message, in case of an emergency.

Recommendation

We recommend implementing a pause mechanism in the CosmWasm `gateway` contract to pause critical functionality in case of an emergency.

Status: Resolved

43. The community pause mechanism can be abused to grief the Solidity `asset-forwarder` contract

Severity: Minor

In `asset-forwarder-contracts:evm/src/AssetForwarder.sol:439-457`, the `communityPause` function of the Solidity `asset-forwarder` contract allows anyone to pause the contract in case of an emergency by staking a certain, predefined amount of native tokens. This functionality can be enabled and disabled by the contract admin via the `isCommunityPauseEnabled` variable.

However, if this pausing mechanism is enabled, anyone with sufficient funds can pause the contract, potentially causing a Denial-of-Service (DoS).

As a pauser is not able to withdraw the staked funds themselves, instead having to rely on the contract admin to refund the staked funds, this will most likely prevent malicious abuse.

Nevertheless, it is a potential risk, and the required staking amount, defined by the `pauseStakeAmountMin` and `pauseStakeAmountMax` range, should be carefully chosen by setting it to a high enough value.

Recommendation

We recommend setting the `pauseStakeAmountMin` and `pauseStakeAmountMax` values reasonably high. Alternatively, consider implementing a list of allowed addresses that are able to pause the contract.

Status: Acknowledged

44. Using Solidity's `transfer` function may prevent relaying funds to the destination chain

Severity: Minor

The `iRelay` function in `asset-forwarder-contracts:evm/src/AssetForwarder.sol:287-329` is called by a forwarder to relay funds to the depositor on the destination chain. If the depositor wishes to receive the funds as wrapped native tokens, the forwarder must send native tokens along with the iRelay call before sending them to the depositor in line 314.

However, the native tokens are sent via Solidity's transfer function, which only forwards a gas stipend of 2300 to the recipient address, leading to potential issues when the recipient is a contract. Specifically, the transfer will fail when the contract's receive or payable fallback function consumes more than the forwarded 2300 gas units. Consequently, the funds can not be relayed to the destination chain, and the depositor has to create a refund on the source chain.

Moreover, this behavior differs from the iRelayMessage, which sends wrapped native tokens to the depositor.

Recommendation

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

Status: Resolved

45. Fees are not validated to be below 100%

Severity: Minor

In several instances of the middleware codebase, fee mechanisms are applied, but their value is not validated to be below 100%.

Firstly, the GAS_FACTOR state is not validated to be higher than 100 and lower than 1000 in asset-forwarder-middleware:contracts/asset-forwarder/src/contract.rs:35 during contract instantiation and only validated at a later point in time when the gas factor is updated via the set_gas_factor setter function.

Secondly, during contract instantiation, the STATIC_FEE state is not validated to be lower than 100 in asset-forwarder-middleware:contracts/asset-forwarder/src/contract.rs:38. This is required in asset-forwarder-middleware:contracts/asset-forwarder/src/queries.rs:485 when computing the system fee so the fee percentage taken is less than 100%.

Lastly, the PARTNER_FEE_IN_BPS is not validated to be lower than 10000 in asset-forwarder-middleware:contracts/asset-forwarder/src/contract.rs:1017. This is required in asset-forwarder-middleware:contracts/asset-forwarder/src/queries.rs:474 when computing the partner fee, so the fee percentage is less than 100%.

Recommendation

We recommend validating the fee amount mentioned above.

Status: Resolved

46. NEAR contracts' initialization process can be front-run

Severity: Minor

In `asset-forwarder-contracts:near/contracts/asset-forwarder/src/contract.rs:51`, the `new` function used to instantiate the NEAR `asset-forwarder` 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 `asset-forwarder-contracts:near/contracts/message-handler/src/lib.rs:30` and `asset-forwarder-contracts: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-forwarder`, message handler, and token contract initialization phase.

Status: Resolved

47. Minimum and maximum pause staked amount is not validated

Severity: Minor

In `asset-forwarder-contracts:evm/src/AssetForwarder.sol:91-92`, the `update` function does not validate the values of `minPauseStakeAmount` and `maxPauseStakeAmount` to be correct. Specifically, the `minPauseStakeAmount` value should be lesser or equal to the `maxPauseStakeAmount` or vice versa.

Consequently, misconfiguring the `pauseStakeAmountMin` and `pauseStakeAmountMax` values would cause lines `446-450` to fail because the stake amount will be out of range, preventing the community from pausing the contract.

Recommendation

We recommend validating minPauseStakeAmount to be lesser or equal to maxPauseStakeAmount.

Status: Resolved

48. The recipient will receive the contract’s NEAR balance instead of the specified amount

Severity: Minor

In asset-forwarder-contracts:near/contracts/asset-forwarder/src/execution.rs:836, 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

49. Incomplete fee mechanism and system fee withdrawal

Severity: Minor

In several instances of the codebase, the implemented fee mechanism is incomplete.

Firstly, system fees are not charged and deducted properly when withdrawing blocked funds and refunding funds in asset-forwarder-middleware:contracts/asset-forwarder/src/execution.rs:130 and line 258. The system fee should be deducted from the fund amount and credited to the middleware contract.

Secondly, there is no entry point for the owner to withdraw the system fee charged in the contract. In lines 842 and 880, the system fee is charged and credited to the middleware contract, but there is no entry point for the owner to withdraw them.

Lastly, the comments in lines 470, 846, and 884 indicate that the partner fees are not implemented fully. Currently, partners cannot receive any fees or incentives. The partner fee should be computed using the fetch_partner_fee_in_bps function and withdrawn to their chosen address.

Recommendation

We recommend completing the fee mechanism mentioned above.

Status: Resolved

50. Incompatibility of deposit ID integer types

Severity: Minor

In asset-forwarder-middleware:contracts/asset-forwarder/src/msg.rs:75 and 99, the HandleFundDeposit and HandleDepositInfoUpdate enums specify the deposit_id integer as u64 value. This is problematic because the Solidity and NEAR asset-forwarder contract specifies them as uint256 and U128 integer types in asset-forwarder-contracts:evm/src/AssetForwarder.sol:30 and asset-forwarder-contracts:near/contracts/asset-forwarder/src/contract.rs:25.

Consequently, the middleware contract cannot handle cross-chain requests with the deposit_id values larger than u64::MAX, causing cross-chain requests to fail.

Recommendation

We recommend modifying the deposit_id integer value to use the u256 integer type.

Status: Acknowledged

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

Severity: Minor

In asset-forwarder-middleware:contracts/asset-forwarder/src/queries.rs:40, the fetch_token_mapping_config function performs an unbounded iteration over the TOKEN_TO_SYMBOL_MAPPING storage to get the values. If there are too many storage entries to process, the query will fail due to an out-of-gas error.

This issue also affects the following queries:

- fetch_all_white_listed_contracts function in line 42
- fetch_all_default_white_listed_contracts function in line 44
- all_pause_info function in line 51
- fetch_all_chain_bytes_info function in line 58

- `fetch_claimable_amounts` function in line `96`
- `fetch_tokens_config` function in line `120`
- `fetch_partners_bps_config` function in line `121`
- `fetch_blocked_fp_requests` function in line `136`

Recommendation

We recommend implementing a pagination mechanism to support batch queries.

Status: Partially Resolved

52. `FetchBalance` query message will always return zero `ROUTE` tokens

Severity: Minor

In `asset-forwarder-middleware:contracts/asset-forwarder/src/queries.rs:226`, the `fetch_balance` function queries the contract's `ROUTE` tokens balance with the denom specified as uppercase "ROUTE" denom. This is incorrect because `ROUTE` tokens are denominated in lowercase "route" denom, as seen in `asset-forwarder-middleware:contracts/asset-forwarder/src/modifiers.rs:68`.

Consequently, the `FetchBalance` query message will always return a zero balance amount, which is incorrect.

Recommendation

We recommend modifying line `226` to use the lowercase "route" denom to query the contract's `ROUTE` token balance correctly.

Status: Resolved

53. Inconsistent state in case of an error during the `fund_deposit_post_processing` function call, resulting in the depositor receiving the funds twice

Severity: Minor

In `asset-forwarder-middleware:contracts/asset-forwarder/src/execution.rs:365-380`, the `handle_fund_deposit` function temporarily puts the deposited funds in the `BLOCKED_FUNDS` storage state for the depositor to withdraw later if the `fund_deposit_post_processing` function called in line `345` errored.

However, any storage writes during the fund_deposit_post_processing function call will not get rolled back in case of an error, resulting in an inconsistent storage state. Specifically, the MESSAGE_HASH_DATA and MESSAGE_HASH_BY_DEPOSIT_INFO storage states are stored in lines 520-529, which will persist in storage if the subsequent fund_paid_post_processing function calls errors. This is problematic because the depositor can initiate a refund using the CreateRefundRequest message, as seen in line 222.

Consequently, the depositor can receive twice the deposited funds by calling the WithdrawBlockedFunds and CreateRefundRequest messages, causing a loss of funds scenario.

While it seems unlikely that such an error occurs, it is recommended to revert any storage writes up until this point to ensure the state is always consistent.

Recommendation

We recommend reverting any storage writes from the fund_deposit_post_processing function call in case of an error. This can be achieved using a submessage as ReplyOn::Error and adding the deposited funds to the BLOCKED_FUNDS storage state within the reply function.

Status: Resolved

54. Incorrect errors in NEAR asset-forwarder contract

Severity: Informational

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

- asset-forwarder-contracts:near/contracts/asset-forwarder/src/execution.rs:126, 149, 157: i_send should be i_deposit.
- asset-forwarder-contracts:near/contracts/asset-forwarder/src/execution.rs:203, 211: i_send should be i_deposit_with_message.
- asset-forwarder-contracts:near/contracts/asset-forwarder/src/execution.rs:243, 258: i_send should be ideposit_info_update.
- asset-forwarder-contracts:near/contracts/asset-forwarder/src/execution.rs:327: i_send should be i_relay.
- asset-forwarder-contracts:near/contracts/asset-forwarder/src/execution.rs:430, 468, 477, 495: i_relay should be i_relay_with_message.
- asset-forwarder-contracts:near/contracts/asset-forwarder/src/execution.rs:598: i_receive should be handle_message_callback.
- asset-forwarder-contracts:near/contracts/asset-forwarder/src/setters.rs:28: set_router_middleware_base should be set_gateway.
- asset-forwarder-contracts:near/contracts/asset-forwarder/src/setters.rs:138, 143: grant_role should be revoke_role.

- asset-forwarder-contracts:near/contracts/asset-forwarder/src/utils.rs:126, 135, 142, 148, 159, 165, 174, 181, 187: i_relay should be irelay_with_message.
- asset-forwarder-contracts:near/contracts/asset-forwarder/src/utils.rs:220: src_token should be included in the error string (see line 213).
- asset-forwarder-contracts:near/contracts/asset-forwarder/src/utils.rs:275: src_token and message should be included in the error string (see lines 267 and 269).
- asset-forwarder-contracts:near/contracts/asset-forwarder/src/utils.rs:332: the error string should only include request_type, src_token, deposit_id, and initiate_withdrawal.

Recommendation

We recommend correcting the errors mentioned above.

Status: Resolved

55. 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.
2. The new owner account claims ownership, which applies the configuration changes.

Status: Acknowledged

56. Hardcoded packet version for cross-chain requests

Severity: Informational

In asset-forwarder-middleware:contracts/asset-forwarder/src/execution.rs:198, 303, and 697, a hardcoded version is used when constructing the cross-chain call packet for different withdrawal scenarios, such as withdrawing blocked funds or refund requests. This may cause inconsistency and potential issues in the future when it is processed by the counterparty chain. For example, the counterparty chain might reject the request due to version mismatches if a different packet version is accepted instead.

Recommendation

We recommend implementing a mechanism to query Router Chain for the latest packet version and use it when initiating a cross-chain request.

Status: Acknowledged

57. Dead code in `handle_deposit_info_update` function

Severity: Informational

In `asset-forwarder-middleware:contracts/asset-forwarder/src/sudo.rs:155`, the `fetch_token_info` function is called without using its return value. Additionally, since it is executed as `unwrap_or_default`, any error that occurs will cause the function to return its default values, causing its execution to be unnecessary.

Recommendation

We recommend removing the code as it is not used anywhere in the codebase.

Status: Acknowledged

58. Missing emission of message hash attribute

Severity: Informational

In `asset-forwarder-middleware:contracts/asset-forwarder/src/execution.rs:533`, the `fund_deposit_post_processing` function attempts to return the response with the message hash attribute. However, the attribute will not be emitted because the `fund_paid_post_processing` function in line `532` will enter first, which eventually causes the response to return in line `547` or `562`. As a result, the message hash attribute will not be emitted as part of the response.

Recommendation

We recommend adding the message hash attribute in the response field in lines `547` and `562`.

Status: Resolved

59. Emitted source chain ID is invalid

Severity: Informational

In `asset-forwarder-middleware:contracts/asset-forwarder/src/sudo.rs:144`, the `handle_deposit_info_update` function emits the `src_chain_id` attribute as the “src_chain_id” string. This is incorrect because the attribute should reflect its intended value.

Recommendation

We recommend modifying line `144` to emit the `src_chain_id` attribute as its value in line `128`.

Status: Resolved

60. Unnecessary state rollbacks implemented

Severity: Informational

In several instances of the NEAR `asset-forwarder` codebase, state rollbacks are implemented before a panic. As panics will revert to the contract’s original state, manual state rollbacks are unnecessary.

For example, the `i_relay_with_message` function reverts the reentrancy lock when the message hash cannot be computed in `asset-forwarder-contracts:near/contracts/asset-forwarder/src/execution.rs:457`, then panics in line `462` with a custom error. In this case, the unlock is unnecessary as the panic will revert the transaction state, which includes the reentrancy lock.

Recommendation

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

Status: Resolved

61. Unneeded payable annotations

Severity: Informational

In `asset-forwarder-contracts:near/contracts/asset-forwarder/src/execution.rs:632`, 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-forwarder contract.

This issue also exists in line 761, where the forwarder_withdraw_refund function implements the #[payable] annotation, but no native funds are required nor expected when withdrawing funds.

Recommendation

We recommend removing the #[payable] annotations.

Status: Resolved

62. Inconsistent amount validation for MAX_TRANSFER_SIZE

Severity: Informational

In asset-forwarder-contracts:evm/src/AssetForwarder.sol:213, 287 and 331, the iDepositInfoUpdate, iRelay, and iRelayMessage functions do not validate the amount transacted not to exceed the MAX_TRANSFER_SIZE. This is inconsistent with the NEAR asset-forwarder contract implementation as the amount is validated in asset-forwarder-contracts:near/contracts/asset-forwarder/src/execution.rs:42 before calling the i_deposit_with_message, i_relay, and i_relay_with_message functions in lines 77, 102, and 108.

Recommendation

We recommend validating the MAX_TRANSFER_SIZE in evm/src/AssetForwarder.sol for the iDepositInfoUpdate, iRelay, and iRelayMessage functions.

Status: Resolved

63. Inconsistent pause modifier for the iReceive function

Severity: Informational

In asset-forwarder-contracts:evm/src/AssetForwarder.sol:388, the iReceive function does not check if the contract is paused using the whenNotPaused modifier. This is inconsistent with the NEAR asset-forwarder contract implementation as the pause validation is performed in asset-forwarder-contracts:near/contracts/asset-forwarder/src/execution.rs:639 in the i_receive function.

Recommendation

We recommend adding the `whenNotPaused` modifier to the `iReceive` function.

Status: Acknowledged

64. Usage of deprecated functionality

Severity: Informational

In `asset-forwarder-contracts:evm/src/AssetForwarder.sol:158`, the `iDepositUSDC` function uses the `safeApprove` function to approve the token messenger for using the USDC in the contract. As the `safeApprove` function is deprecated, its usage is discouraged.

Recommendation

We recommend using the `safeIncreaseAllowance` function instead.

Status: Resolved

65. Fees are not charged for `SetDappMetadata` and `ISend` messages

Severity: Informational

In `cosmwasm-gateway-contract:src/contract.rs:820` and `920`, the `isend` and `set_dapp_metadata` functions do not charge any fees. This is inconsistent with the Solidity and NEAR `gateway` contract, as `iSendDefaultFee` is charged during the `iSend` and `setDappMetadata` functions.

Recommendation

We recommend implementing the fee functionality in the CosmWasm `gateway` contract.

Status: Resolved

66. Vault address is not validated

Severity: Informational

In `cosmwasm-gateway-contract:src/contract.rs:989`, the `set_vault` function stores the provided `vault_contract_address` argument into the `VAULT` storage state without validating the address. If an invalid address is provided, the CosmWasm `gateway` contract will fail to mint and burn tokens correctly.

Recommendation

We recommend validating the address with the `addr_validate` function.

Status: Resolved

67. Nonce instantiation in CosmWasm `gateway` contract differs from other implementations

Severity: Informational

In `cosmwasm-gateway-contract:src/contract.rs:58`, the CosmWasm `gateway` contract instantiates the nonce as 0. This is inconsistent with the Solidity and NEAR `gateway` contract implementation as nonce instantiates as 1.

Recommendation

We recommend instantiating the nonce as 1 for consistency.

Status: Resolved

68. Unneeded custom reply identifiers for `ReplyOn::Never` submessages

Severity: Informational

In several instances of the codebase, custom reply identifiers are set for `ReplyOn::Never` submessages:

- `cosmwasm-gateway-contract:src/contract.rs:696`
- `cosmwasm-gateway-contract:src/contract.rs:721`
- `cosmwasm-gateway-contract:src/contract.rs:843`
- `cosmwasm-gateway-contract:src/contract.rs:896`
- `cosmwasm-gateway-contract:src/contract.rs:962`

This is unneeded because `ReplyOn::Never` submessages are essentially normal messages, and the supplied reply identifier will not be entered regardless of the execution result.

Recommendation

We recommend removing custom identifiers for `ReplyOn::Never` submessages and dispatching them like a normal message to increase code readability.

Status: Resolved

69. Storage costs in NEAR contracts are borne by the deployer

Severity: Informational

In NEAR smart contracts, the deployer is responsible for the storage costs associated with the contract based on the storage staking mechanism. This creates a potential vulnerability where users can exploit the imbalance between the low cost of sending data and the significantly higher cost borne by the contract owner for storing that data, also known as the million cheap data additions attack.

For instance, the `i_relay` function in `asset-forwarder-contracts:near/contracts/asset-forwarder/src/execution.rs:320` increases the storage cost when a new message hash is set in the `execute_record` unordered map. However, the caller is not required to contribute funds to cover this additional storage cost.

Recommendation

We recommend introducing a fee allowance mechanism for users to deposit fees in advance for themselves or others. The deposited funds can be used to cover any additional storage costs incurred, with the excess being withdrawable.

Status: Acknowledged