Audit Report

Timewave Covenants

v1.0

May 2, 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

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

Introduction

Purpose of This Report

Oak Security has been engaged by Timewave Inc. to perform a security audit of Timewave Covenants.

The objectives of the audit are as follows:

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

This report represents a summary of the findings.

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

Codebase Submitted for the Audit

The audit has been performed on the following target:

Repository	https://github.com/timewave-computer/covenants
Commit	`88936e78ae67760f9c7216cbe366fdbcda2e6812`

Scope	The following contracts and packages are in scope:
	
	`.`
	`├── contracts`
	`│  ├── astroport-liquid-pooler`
	`│  ├── ibc-forwarder`
	`│  ├── interchain-router`
	`│  ├── native-router`
	`│  ├── native-splitter`
	`│  ├── osmo-liquid-pooler`

	<pre>│ ├── outpost-osmo-liquid-pooler
	│ ├── remote-chain-splitter
	│ ├── single-party-pol-covenant
	│ ├── single-party-pol-holder
	│ ├── stride-liquid-staker
	│ ├── swap-covenant
	│ ├── swap-holder
	│ ├── two-party-pol-covenant
	│ └── two-party-pol-holder
	└── packages
	    ├── covenant-macros
	    └── covenant-utils</pre>

Fixes verified at commit	`49357643880ce05c79e38c32e4d4cf0f3b38509f` Note that changes to the codebase beyond fixes after the initial audit have not been in scope of our fixes review.

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

Timewave Covenants are smart contracts designed to be deployed on Neutron that enable interchain agreements.

Three types of covenants are implemented:

- Two parties covenant to swap native or interchain tokens.
- Two parties covenant to co-own a liquidity position on Astroport or Osmosis.
- Single-party covenant to natively liquid stake tokens and participate in a liquidity pool that includes both native and liquid staked tokens of the same chain.

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 protocol encompasses several contracts that communicate with each other through message exchanges and callbacks, in addition to utilizing IBC messages and ICA accounts. It also integrates with third-party protocols such as Polytone, Astroport, and Osmosis.
Code readability and clarity	Medium-High	-
Level of documentation	Medium-High	The client provided detailed documentation and diagrams
Test coverage	~~Low~~ Medium-High	`cargo tarpaulin` reports a `10.44%` test coverage at the audited commit. An `interchaintest` test suite is implemented. The client improved the test coverage during issue remediation to `75.11%`.

Summary of Findings

No	Description	Severity	Status
1	Interchain parties cannot be refunded after swap covenant expiration	Critical	Resolved
2	Disregarded emergency committee configuration hinders emergency message execution	Major	Resolved
3	Improper assertions will block migration or allow incorrect configurations to be stored	Major	Resolved
4	Funds stuck in the `native-router` contract due to unnecessary IBC fee reservations	Major	Resolved
5	Neutron governance updating IBC fees could hinder interchain transactions	Major	Resolved
6	Attackers can drain `NTRN` from `interchain-router` contracts by executing `DistributeFallback` messages targeting unsupported coins	Major	Resolved
7	Covenant contract parameter update poses centralization risks	Major	Acknowledged
8	Single-sided liquidity provision is not restricted to stablecoin pairs	Major	Resolved
9	Locked funds in the liquid pooler contract if the liquidity pool is not initialized or outside of the required range	Major	Resolved
10	Lack of verification of matching denominations between `native-splitter` splits and `swap-holder` contributions could lead to stuck funds	Major	Resolved
11	Side based covenant implementation differs from specification	Major	Acknowledged
12	Emergency withdrawal can lead to stuck funds	Major	Resolved
13	Attackers can spam `Tick` messages to disable the execution of withdrawals in the `osmo-liquid-pooler`	Major	Resolved
13	Decimal rounding strategy in `remote-chain-splitter` contract could lead	Minor	Partially Resolved

	to funds stuck in `ibc-forwarder`		
14	Partial covenant configuration validation in the `two-party-pol-holder` contract	Minor	Resolved
15	Lack of grouped versioning of code IDs could lead to malfunctioning covenant deployments	Minor	Acknowledged
16	The `two-party-pol-holder` does not refund excess contributions	Minor	Acknowledged
17	Missing address validation during the `interchain-router contract` instantiation	Minor	Resolved
18	Missing address validation during the `native-splitter` contract instantiation	Minor	Resolved
19	Missing address validation for `parties_config` during the `swap-holder` contract instantiation	Minor	Resolved
20	Asymmetric ragequit penalty for share-based covenants	Minor	Acknowledged
21	Missing `percent` value validation	Minor	Resolved
22	Missing `lockup_period` validation	Minor	Resolved
23	The `stride-liquid-staker` contract allows config updates that may interfere with existing ICA	Minor	Acknowledged
24	Impossibility to recover additional funds from the `ibc-forwarder` and `remote-chain-splitter` contracts' ICA accounts	Minor	Resolved
25	Missing validations in the `native-splitter` contract	Minor	Resolved
26	Migrate entrypoints cannot update `CONTRACT_CODES`	Minor	Resolved
27	Inconsistency in `two-party-pol-holder` contract state transitions	Minor	Resolved
28	Unrelayed ICA transactions could make the `ibc-forwarder` contract unable to redirect funds to holder contracts	Minor	Resolved
29	Inconsistency in `remote-chain-splitter` contract state transitions	Informational	Resolved
30	Missing check to ensure that the `lockup_config` expiration is after the `deposit_deadline`	Informational	Resolved

31	Redundant functions to validate the `clock` address	Informational	Resolved
32	Emergency withdrawal can be executed multiple times	Informational	Resolved
33	Inefficiency in pair validation query	Informational	Resolved
34	Missing split validations in `two-party-pol-holder` migration	Informational	Resolved
35	Unnecessary computation of `ibc-forwarder` addresses in case the party is `Native`	Informational	Resolved
36	“Migrate only if newer” pattern is not followed	Informational	Acknowledged
37	The `ibc-forwarder` contract never reaches the `Complete` state	Informational	Resolved
38	Contract parameter update pattern using migrations is inefficient and error-prone	Informational	Acknowledged
39	Redundant check for duplicates in a loop	Informational	Acknowledged
40	Inefficiencies in iterations	Informational	Acknowledged
41	Code duplication decreases readability and maintainability	Informational	Resolved
42	Remove commented code blocks	Informational	Resolved
43	Remove debug messages	Informational	Resolved
44	Remove dead code	Informational	Resolved
45	Resolve TODO code and comments	Informational	Resolved

Detailed Findings

1. Interchain parties cannot be refunded after swap covenant expiration

Severity: Critical

The `try_refund` function of the `swap-holder` contract, defined in `contracts/swap-holder/src/contract.rs:144-193`, executes the refund to parties in case the covenant is expired.

Refund messages are constructed by the `get_refund_msg` function defined in `packages/covenant-utils/src/lib.rs:93-117`. In case the `receiver_config` type is `ReceiverConfig::Ibc`, the refund message is constructed as an `IbcMsg::Transfer` message.

However, this type of message does not support the `feeRefunder` Neutron Cosmos SDK module, causing an error due to the missing IBC fee payment and reverting the transaction.

As a consequence, it would be not possible to refund parties and funds would be stuck in the contract.

Recommendation

We recommend substituting the `IbcMsg::Transfer` with a `NeutronMsg::IbcTransfer` message and specifying IBC fees.

Status: Resolved

2. Disregarded emergency committee configuration hinders emergency message execution

Severity: Major

In `contracts/two-party-pol-holder/src/contract.rs:34`, the `instantiate` function of the `two-party-pol-holder` contract receives the `emergency_committee_addr` address as a parameter of the `InstantiateMsg`.

However, the `emergency_committee_addr` is not saved in the `EMERGENCY_COMMITTEE_ADDR Item` storage.

As a result, the emergency committee configuration is discarded and hence it is not possible for the committee to execute emergency privileged messages like `EmergencyWithdraw`.

Recommendation

We recommend validating and storing the `emergency_committee_addr` in the `EMERGENCY_COMMITTEE_ADDR` storage `Item`.

Status: Resolved

3. Improper assertions will block migration or allow incorrect configurations to be stored

Severity: Major

In the `migrate` function, defined in `contracts/single-party-pol-holder/src/contract.rs:185-195`, the contract incorrectly ensures that the provided lockup is expired.

As a result, expired states will successfully pass and allow for migration to occur while valid non-expired states will return incorrect `LockupPeriodIsExpired` and `MustBeFutureLockupPeriod` errors.

Recommendation

We recommend modifying the aforementioned assertion to ensure that it returns errors only for expired states.

Status: Resolved

4. Funds stuck in the `native-router` contract due to unnecessary IBC fee reservations

Severity: Major

In `contracts/native-router/src/contract.rs:157-180`, the `try_route_balances` function of the `native-router` contract reserves a portion of `NTRN` coins to pay for IBC fees.

However, it should not reserve IBC fees as the contract is designed for native transfers without IBC interactions.

As a consequence, this leads to a portion of the `NTRN` coins being unnecessarily stuck in the contract.

Recommendation

We recommend modifying the `try_route_balances` function to not reserve `NTRN` coins to pay for IBC fees.

Status: Resolved

5. Neutron governance updating IBC fees could hinder interchain transactions

Severity: Major

In `packages/covenant-utils/src/lib.rs:210-212`, the `get_default_ibc_fee_requirement` function calculates the sum of all the IBC transfer fees required by the Neutron `feeRefunder` Cosmos SDK module relying on hardcoded fee values.

Similarly, in `contracts/ibc-forwarder/src/contract.rs:101-109`, the `try_register_ica` function attempts to register a new ICA account, paying a hardcoded `register_fee`.

Moreover, in `packages/covenant-utils/src/lib.rs:242-249`, the `get_ibc_transfer_messages_for_coins` function assumes that the IBC fees denom is `untrn`.

Additionally, In `contracts/stride-liquid-staker/src/contract.rs:106` the ICA `register_fee` is hard-coded.

However, since Neutron's governance can update the required IBC fees, hardcoded fees could render contracts unable to execute transfers or interchain transactions if fees change.

As a consequence, the aforementioned contracts may become stuck and not able to execute interchain transactions.

Recommendation

We recommend querying the current IBC fees directly from the `feeRefunder` and `interchaintxs` Cosmos SDK modules instead of hardcoding them.

Status: Resolved

6. Attackers can drain `NTRN` from `interchain-router` contracts by executing `DistributeFallback` messages targeting unsupported coins

Severity: Major

In `contracts/interchain-router/src/contract.rs:67-99`, the `try_distribute_fallback` method allows the `interchain-router` contract to permissionlessly distribute coins in its balance with a denom different from the targeted ones.

However, since it is required to pay IBC fees in `NTRN` to send IBC transfer transactions, attackers could send worthless coins to the contract and then execute `DistributeFallback` messages to force the contract to spend `NTRN` on fees for these transactions.

As a consequence, attackers can repeatedly exploit this vulnerability with unsupported denoms, draining all the `NTRN` funds in the contract.

Recommendation

We recommend requiring the `DistributeFallback` message sender to provide funds to pay for IBC fees.

Status: Resolved

7. Covenant contract parameter update poses centralization risks

Severity: Major

The `UpdateConfig` migrate message in `contracts/two-party-pol-holder/src/contract.rs:580` and `contracts/osmo-liquid-pooler/src/contract.rs:740` allows the contract admin to update various critical parameters of the covenant contract.

This poses a serious threat if the admin can make parameter updates after a covenant has been stated and both parties have begun the process. This is problematic because the admin can pass a completely new `covenant_config` which would overwrite the existing configuration and could change multiple aspects of the original agreement that both parties agreed upon. This functionality even allows for the original parties to be replaced, the covenant shares could be changed, etc. None of this should occur after the contract is in an active state.

Additionally, other parameters can be changed without proper validation. For example, the `ragequit_config` can be updated to include a penalty value that is greater than the share of one of the parties, which would effectively block that party from being able to rage quit due to a subtraction panic. Similarly, it does not prevent the `lp_config` from being updated even if the contract is in an active state.

Recommendation

We recommend only allowing these critical configuration updates if the contract is in the `ContractState::Instantiated` state.

Status: Acknowledged

The client states they expect the administration of the covenants to be managed through a 2-of-2 multi-signature or a DAO DAO deployment, involving both parties. This arrangement would enable them to modify the covenant terms via suitable governance mechanisms. Furthermore, in scenarios where the parties operate on a foreign chain, the client mentioned the possibility of establishing an interchain account or a Polytone proxy account first.

8. Single-sided liquidity provision is not restricted to stablecoin pairs

Severity: Major

In `contracts/astroport-liquid-pooler/src/contract.rs:247-256`, single-sided liquidity provision is attempted when the user has a positive balance for only one of the two tokens.

However, Astroport user documentation states that single-sided LP is possible only for stableswap pairs. With other token pairs, single-sided LP will shift the pool ratio.

Recommendation

We recommend restricting single-sided liquidity provision to stableswap pairs only.

Status: Resolved

9. Locked funds in the liquid pooler contract if the liquidity pool is not initialized or outside of the required range

Severity: Major

In `contracts/astroport-liquid-pooler/src/contract.rs:229`, the pool ratio `a_to_b_ratio` is calculated and checked to match the preconfigured allowed range.

However, there are two concerns:

- The following expression: `Decimal::from_ratio(pool_token_a_bal, pool_token_b_bal)` cannot be computed when `pool_token_b_bal` is zero since it would cause a division by zero error. In other words, the liquidity provision transaction is rejected if the liquidity has not been provided yet.

- In case the pool ratio is outside of the required bounds it would not be possible to provide the tokens since the execution would error and revert the transaction.

Consequently, since the `Withdraw` message is able to redeem funds only from the pool and not from the contract balance, funds could be stuck in the contract if the pool is not initialized yet or the pool ratio is outside of the specified range.

Recommendation

We recommend handling the zero liquidity amount case, allowing the `Withdraw` message to redeem funds in the contract balance and specifying different pool ratio bounds for the time when the pool is in specific states.

Although the same level of risk may not impact the `osmo-liquid-pooler`, we advise applying the same recommendation to this contract.

Status: Resolved

10. Lack of verification of matching denominations between `native-splitter` `splits` and `swap-holder` `contributions` could lead to stuck funds

Severity: Major

In `contracts/swap-covenant/src/contract.rs:124-171`, the `instantiate` function of the `swap-covenant` constructs and executes the `InstantiateMsg` for the `native-splitter` and `swap-holder` contracts.

The `native-splitter` is configured to distribute funds depending on the information stored in `splits` and the `swap-holder` is configured to handle funds defined in `party_x_config.contribution`.

However, it is not verified that those two sources contain information about the same denoms and with the same total amount of coins.

As a consequence, in case the provided values refer to different coins, the `native-splitter` would not be able to complete the swap since the required funds are not provided by the `swap-holder`.

Recommendation

We recommend validating in the `swap-covenant` contract that `splits` and `party_x_config.contribution` refer to the same coins.

Status: Resolved

11. Side based covenant implementation differs from specification

Severity: Major

Side based covenants are defined in the specification as follows:

"Each party has a claim to their side (denom) of the liquidity. E.g. if Cosmos Hub (ATOM) and Neutron (NTRN) are the two parties joining the pool, Cosmos Hub has a claim to all the ATOM associated with the position and Neutron has a claim to all the NTRN associated with the position."

However, the implementation differs from the specification since the `try_claim_side_based` function, defined in `contracts/two-party-pol-holder/src/contract.rs:331-352`, operates by distributing the retrieved tokens based on the provided `splits` and `shares` which are arbitrarily set by the covenant instantiator.

As a consequence, this discrepancy leads to users receiving `shares` calculated on `splits` instead of solely their allocated side of the pool.

Recommendation

We recommend revising the `try_claim_side_based` function to ensure users receive only their side of the pool during `Side` covenants `Claim` events.

Status: Acknowledged

The client states that the configuration options are deliberately designed with high flexibility to support innovative use cases and experimentation. The client also acknowledges that while some configurations might be less practical, maintaining this high level of flexibility is a choice meant to foster creativity and adaptability as the product evolves.

12. Emergency withdrawal can lead to stuck funds

Severity: Major

The `try_emergency_withdraw` function, defined in `contracts/single-party-pol-holder/src/contract.rs:102-119`, allows the `EMERGENCY_COMMITTEE_ADDR` to execute an emergency withdrawal.

However, during the process, the `WITHDRAW_TO` storage variable is not validated.

Consequently, in cases when `WITHDRAW_TO` is not set, the consecutive call to the `try_distribute` function from the pooler back to the holder will fail leading to the funds being stuck in the pooler.

We report this issue with major severity since it can be remediated by the admin who would need to plan, craft the correct state and then execute the `migrate` function to complete the withdrawal request.

Recommendation

We recommend validating all state variables necessary for the execution of `try_distribute` and refactoring code parts shared between `try_emergency_withdraw` and `try_claim` functions to deduplicate the code. Code deduplication can increase maintainability and security, specifically, the `try_emergency_withdraw` function is similar to the `try_claim` function. While the correct validation of `WITHDRAW_TO` has been implemented in the `try_claim` function, it has not been applied to the `try_emergency_withdraw` function.

Status: Resolved

13. Attackers can spam `Tick` messages to disable the execution of withdrawals in the `osmo-liquid-pooler` contract

Severity: Major

In `contracts/osmo-liquid-pooler/src/contract.rs:417-437`, the `try_sync_proxy_balances` function of the `osmo-liquid-pooler` contract queries the proxy for balances.

However, since the `reset_latest_proxy_balances` function removes old balances stored in the contract in line `433`, the contract does not have information about the proxy balances in the duration between the execution of this transaction and the execution of the response handler.

Attackers could spam `tick` messages targeting the `try_sync_proxy_balances` function in the same block as `Withdraw` transactions to grieve holders. Depending on the transaction ordering, if the `Tick` message is executed before the `Withdraw` message, the withdrawal will fail.

Recommendation

We recommend implementing an intermediate contract state to prevent attackers from repeatedly executing the `try_sync_proxy_balances` function.

Status: Resolved

14. Decimal rounding strategy in `remote-chain-splitter` contract could lead to funds stuck in `ibc-forwarder`

Severity: Minor

In `contracts/remote-chain-splitter/src/contract.rs:167-172`, the `try_split_funds` function of the `remote-chain-splitter` contract iterates through all the `splits` to compute the amount to send to each `receiver` address depending on its `share`.

However, since this computation is performed by multiplying the total amount by the `share` of the respective `receiver` using the `checked_multiply_ratio` function, it always rounds down the result, which could lead to the loss of one base unit of the coin.

Those funds are then sent to the configured `ibc-forwarder` contracts which rely on the amount provided by the `single-party-pol-covenant`, manually computed by the covenant instantiator in `config.contribution`.

Consequently, if the covenant instantiator did not account for the decimal rounding when calculating the `contribution`, the `ibc-forwarder` would always fail to perform the IBC transfer from the ICA account and a manual intervention would be required.

For instance, if the amount is 100 and the shares are 0.107 and 0.893, it will result in sending 10 and 89 coins to the two receivers, leading to a total of 99 coins and a loss of 1 coin. If the covenant instantiators defined `contribution` equal to 11 and 89, the first `ibc-forwarder` contract will be unable to forward coins to the recipient.

Recommendation

We recommend adjusting the decimal rounding strategy and ensuring in the `single-party-pol-covenant` that the sum of the amounts in the provided `contribution` and `splits` are equal.

Status: Partially Resolved

15. Partial covenant configuration validation in the `two-party-pol-holder` contract

Severity: Minor

During the instantiation of the `two-party-pol-holder` contract, in `contracts/two-party-pol-holder/src/msg.rs:275-282`, the `validate` method checks the correctness of the covenant configuration.

However, it fails to ensure for each party that the `host_addr` represents a valid address and that `contribution` is a valid coin.

Recommendation

We recommend ensuring that `host_addr` is a valid address and that `contribution` is a valid coin in the `validate` method.

Status: Resolved

16. Lack of grouped versioning of code IDs could lead to malfunctioning covenant deployments

Severity: Minor

Covenant contracts enable instantiators to provide a specific code ID to each actor to be instantiated.

However, due to the possibility of having several code IDs per contract, creators might assign code IDs that are incompatible because they are intended for a different version of the covenant.

Additionally, it poses a security risk, as there is a risk of users supplying incorrect or harmful code IDs.

Recommendation

We recommend implementing a grouped versioning for actor contracts and letting instantiators provide the covenant version instead of the specific code IDs.

Status: Acknowledged

The client states they intend to ensure version compatibility by utilizing off-chain methods, including the frontend and documentation.

17. The `two-party-pol-holder` does not refund excess contributions

Severity: Minor

In `contracts/two-party-pol-holder/src/contract.rs:372`, the `try_deposit` function verifies if both parties have deposited the required `contribution` funds defined in the covenant.

However, since it is only checked that the provided funds are greater or equal to the required `contribution`, in case a party deposits more funds than required, the excess will not be refunded.

As a consequence, excess contributions will be stuck in the `two-party-pol-holder` contract.

Recommendation

We recommend implementing refund logic for excess deposits in the `two-party-pol-holder` contract.

Status: Acknowledged

The client states they plan to introduce more complex refund mechanisms in future versions. They acknowledge this issue as a known limitation and advise users to be cautious not to deposit extra funds into the contracts.

18. Missing address validation during the `interchain-router` contract instantiation

Severity: Minor

During the `interchain-router` contract instantiation, in `contracts/interchain-router/src/contract.rs:31-38`, `clock_address` and `destination_config` addresses are stored in the contract without being validated.

As a consequence, incorrect addresses could be stored in the contract leading to the impossibility of the `interchain-router` to perform actions.

Recommendation

We recommend validating `clock_address` and `destination_config` addresses before storing them in the `interchain-router` contract.

Status: Resolved

19. Missing address validation during the `native-splitter` contract instantiation

Severity: Minor

During the `native-splitter` contract instantiation, in `contracts/native-splitter/src/contract.rs:32`, `clock_address` is stored in the contract without being validated.

As a consequence, incorrect addresses could be stored in the contract, leading to the impossibility of the `native-splitter` to perform actions.

Recommendation

We recommend validating the `clock_address` address before storing it in the `native-splitter` contract.

Status: Resolved

20. Missing address validation for `parties_config` during the `swap-holder` contract instantiation

Severity: Minor

During the `swap-holder` contract instantiation, in `contracts/swap-holder/src/contract.rs:42`, `parties_config` containing parties' addresses are stored in the contract without being validated.

As a consequence, incorrect addresses could be stored in the contract leading to the impossibility of the `swap-holder` to perform actions.

Recommendation

We recommend validating `CovenantParty` addresses before storing them in the contract.

Status: Resolved

21. Asymmetric ragequit penalty for share-based covenants

Severity: Minor

The `try_ragequit` function, defined in `contracts/two-party-pol-holder/src/contract.rs:506`, applies a penalty to the rage quitting party.

For covenants that are share-based, the penalty is directly subtracted from the rage quitting party's allocation.

However, this design is potentially problematic because it does not take into account the size of the allocation.

Consequently, this presents a situation where a ragequitting party with a smaller allocation share could lose nearly their entire position whereas the larger party stands to lose a very small proportion of their total allocation.

Recommendation

We recommend implementing a proportional ragequit penalty amount rather than applying a flat penalty.

Status: Acknowledged

22. Missing `percent` value validation

Severity: Minor

The `try_withdraw` function, defined in `contracts/astroport-liquid-pooler/src/contract.rs:111` and `contracts/osmo-liquid-pooler/src/contract.rs:125`, does not validate the `percent` argument.

If `percent` is `None`, then `Decimal::one()` will be the default value. However, the `percent` value could also be equal to 0 or be more than 1, leading to incorrect configurations.

Recommendation

We recommend validating that the `percent` value is more than 0 and less than or equal to 1 according to `contracts/astroport-liquid-pooler/src/contract.rs:112`.

Status: Resolved

23. Missing `lockup_period` validation

Severity: Minor

During the execution of the `instantiate` function defined in `contracts/single-party-pol-holder/src/contract.rs:43`, the `msg.lockup_period` is not validated properly, potentially allowing misconfiguration.

There are multiple scenarios where a misconfigured expiration value can be saved during instantiation. For example, an expiration could be set to a value that is in the past, or it could be set to `Never`.

Recommendation

We recommend validating `msg.lockup_period` before storing it in the contract.

Status: Resolved

24. The `stride-liquid-staker` contract allows config updates that may interfere with existing ICA

Severity: Minor

The `UpdateConfig` migrate message in `contracts/stride-liquid-staker/src/contract.rs:462` allows the admin to update the config parameters of the `stride-liquid-staker` contract.

However, it does not validate the current state before it commits these updates to the contract state.

This can be problematic if the newly updated parameters conflict with the existing state of an active contract. For example, if the contract already has an active interchain account associated with it, updating the `remote_chain_info` could impact the current connection and not enable the contract to communicate with the previously created ICA account.

Recommendation

We recommend ensuring that the contract is in the `Instantiated` state before allowing for the `remote_chain_info` to be updated.

Status: Acknowledged

The client states they decided to prioritize migration flexibility to guarantee recovery from any issues arising in production. They indicate that in future versions of the covenants contracts, they will impose stricter limitations on contract updates.

25. Impossibility to recover additional funds from the `ibc-forwarder` and `remote-chain-splitter` contracts' ICA accounts

Severity: Minor

The `ibc-forwarder` and `remote-chain-splitter` contracts lack functionality for transferring funds with denoms different from the target ones from the ICA account.

As a consequence, in case someone erroneously deposits funds in the ICA account, the deposited funds would be unrecoverable.

Recommendation

We recommend implementing a message similar to the `DistributeFallback` implemented in other contracts in the `ibc-forwarder` and `remote-chain-splitter` contracts.

Status: Resolved

26. Missing validations in the `native-splitter` contract

Severity: Minor

In `contracts/native-splitter/src/contract.rs:32`, during the execution of the `instantiate` function of the `native-splitter` contract, the `clock_address` address is not validated.

Similarly, in `contracts/native-splitter/src/contract.rs:41-47`, `fallback_split` is not validated before being stored in the contract.

Recommendation

We recommend validating `clock_address` and `fallback_split` within the `instantiate` function of the `native-splitter` contract.

Status: Resolved

27. Migrate entrypoints cannot update `CONTRACT_CODES`

Severity: Minor

Migrate entrypoints of contracts within the scope of this audit do not allow for updating the `CONTRACT_CODES`, which are essential for migrating the code IDs of the covenant actors' sub-contracts.

As a consequence, it is not possible with the current version of the migration logic to update `CONTRACT_CODES`, resulting in difficulties for future migration which should handle this in the same transaction.

Recommendation

We recommend implementing a `MigrateMsg` to enable `CONTRACT_CODES` updates, thus facilitating contract ID migrations.

Status: Resolved

28. Inconsistency in `two-party-pol-holder` contract state transitions

Severity: Minor

In `contracts/two-party-pol-holder/src/contract.rs:397-430`, the `try_deposit` function of the `two-party-pol-holder` contract sets the contract state to `Completed` and dequeues the contract from the `clock` if the deposit deadline has expired and no funds are available in the contact.

However, it does not change the contract state if the deposit deadline has expired but there are some funds to send to routers.

As a consequence, the contract remains in the `clock` contract's queue and retains the `Instantiated` state, potentially leading to inconsistencies in contract state transitions.

Additionally, there is a case where the contract enters into the `Completed` state without dequeuing from the `clock` in lines `163`, `318`, and `347` when the parties claimed all of their allocations. This results in continuously handling of `ticks` from the `clock` without producing any actual effect.

Recommendation

We recommend setting the contract state to `Completed` and dequeuing the contract from the `clock` for all the cases related to deposit deadline expiration.

Status: Resolved

29. Unrelayed ICA transactions could make the `ibc-forwarder` contract unable to redirect funds to holder contracts

Severity: Minor

Holder contracts are instantiated with an `Instantiated` state which lets the `tick` method execute the `try_deposit` function.

If the deposit deadline is expired this function refunds the parties and advances the state to `Completed`.

However, in case one of the parties decides to provide funds from another chain leveraging the `ibc-forwarder` contract, and the ICA transaction to move funds from the remote chain to the host chain is not relayed, the holder contract is not able to receive funds because the `Completed` states do not allow the `tick` method to perform any actions.

We report this issue with minor severity since ICA transactions relaying is permissionless and the deposit deadline duration is orders of magnitudes larger than the time to relay the transaction.

Recommendation

We recommend implementing a handler in holder contracts to refund parties even if the state is `Completed`.

Status: Resolved

30. Inconsistency in `remote-chain-splitter` contract state transitions

Severity: Informational

The `Completed` state of the `remote-chain-splitter` is reached only in the `complete_and_dequeue` function in `contracts/remote-chain-splitter/src/msg.rs:96`.

However, since this function is never called, the contract will never reach the `Completed` state.

Additionally, the `try_register_ica` function in `contracts/remote-chain-splitter/src/contract.rs:128` stores empty data in `INTERCHAIN_ACCOUNTS` but does not clear it in `sudo_timeout` in `contracts/remote-chain-splitter/src/sudo.rs:73` when the contract transitions back to the `Instantiated` state.

Recommendation

We recommend allowing the contract state to reach the `Completed` state and clearing the account data.

Status: Resolved

31. Missing check to ensure that the `lockup_config` expiration is after the `deposit_deadline`

Severity: Informational

In `contracts/two-party-pol-holder/src/contract.rs:34-53`, during the execution of the `instantiate` function of the `two-party-pol-holder` contract, it is not checked that the `lockup_config` expiration is after the `deposit_deadline`.

As a consequence, there could be scenarios where, following the deposit, the covenant is already considered expired and parties could immediately claim LP tokens.

Recommendation

We recommend implementing a check in the `instantiate` function of the `two-party-pol-holder` contract to ensure that the `lockup_config` expiration is after the `deposit_deadline`.

Status: Resolved

32. Redundant functions to validate the `clock` address

Severity: Informational

In `contracts/swap-holder/src/contract.rs:67-70`, the address of the clock is validated within `ensure` macros.

However, in all other places, the address is validated using `verify_clock` function defined in `contracts/clock/src/helpers.rs:24`.

Code duplication reduces maintainability and is error-prone, as changes must be replicated across similar code segments, increasing the chance of inconsistency and bugs. Eliminating such duplication improves system reliability and simplicity.

Recommendation

We recommend validating the clock address using the `verify_clock` function in all instances.

Status: Resolved

33. Emergency withdrawal can be executed multiple times

Severity: Informational

The `try_emergency_withdraw` function, defined in `contracts/single-party-pol-holder/src/contract.rs:102-119`, allows the `EMERGENCY_COMMITTEE_ADDR` to execute an emergency withdrawal.

However, during the process, the `WITHDRAW_STATE` is not updated and stored in the contract.

Consequently, it is possible to call the `try_emergency_withdraw` function multiple times or execute multiple withdraw operations like `try_emergency_withdraw` and `try_claim` at the same time bypassing the guards checking the `WITHDRAW_STATE`.

Recommendation

We recommend executing `WITHDRAW_STATE.save()` at the end of the `try_emergency_withdraw` function execution.

Status: Resolved

34. Inefficiency in pair validation query

Severity: Informational

The `try_lp` function, defined in `contracts/astroport-liquid-pooler/src/contract.rs:210`, is called every `tick` when the contract is in the `Instantiated` state.

For every call, the `validate_pair_type` function is invoked. This function queries the pool type and checks its config against the pool config.

However, it is inefficient to execute this query for every invocation of the `try_lp` function as `LP_CONFIG` is a state variable and the Astroport Pair contracts do not support the update of the pool type.

Consequently, this value could be checked during the instantiation rather than dispatching a query to check for each tick.

Recommendation

We recommend checking the pair type in the `astroport-liquid-pooler` contract at the time of instantiation rather than re-querying an immutable value.

Status: Resolved

35. Missing split validations in `two-party-pol-holder` migration

Severity: Informational

In `contracts/two-party-pol-holder/src/contract.rs:637-649`, `splits` and `fallback_split` are not validated during the migration.

However, since they are validated during the contract instantiation in `contracts/two-party-pol-holder/src/contract.rs:66` and `78`, they should be validated also during the migration.

Recommendation

We recommend validating the `splits` and `fallback_split` in the `migrate` method.

Status: Resolved

36. Unnecessary computation of `ibc-forwarder` addresses in case the party is `Native`

Severity: Informational

In `contracts/two-party-pol-covenant/src/contract.rs:79-86`, `contracts/swap-covenant/src/contract.rs:81-92` and `contracts/single-party-pol-covenant/src/contract.rs:68-79` the execution always computes the addresses for both parties's `ibc-forwarder` contracts.

However, this computation is not necessary if `msg.party_a_config` or `msg.party_b_config` are not of the type `CovenantPartyConfig::Interchain`, leading to inefficient computation and a higher transaction gas cost.

Recommendation

We recommend computing `ibc-forwarder` addresses only if the party type is `CovenantPartyConfig::Interchain`.

Status: Resolved

37. "Migrate only if newer" pattern is not followed

Severity: Informational

The contracts within the scope of this audit are currently migrated without regard to their version. This can be improved by adding validation to ensure that the migration is only performed if the supplied version is newer.

Recommendation

We recommend following the "migrate only if newer" pattern defined in the CosmWasm documentation.

Status: Acknowledged

The client states they decided to prioritize migration flexibility to guarantee recovery from any issues arising in production. They indicate that in future versions of the covenants contracts, they will impose stricter limitations on contract updates.

38. The `ibc-forwarder` contract never reaches the `Complete` state

Severity: Informational

The `ibc-forwarder` contract defines a `ContractState::Complete` state and implements a `Tick` handler for it in `contracts/ibc-forwarder/src/contract.rs:94-97`.

However, since there is no transition to it, this state is never reached.

Recommendation

We recommend implementing a transition to the `ContractState::Complete` state, thereby achieving the intended final state of completion.

Status: Resolved

39. Contract parameter update pattern using migrations is inefficient and error-prone

Severity: Informational

The contracts within the scope of this audit allow for parameter updates via a migration entrypoint, specifically through the use of a `MigrateMsg::UpdateConfig` migration message. This method ensures that only the contract `admin` can initiate changes, yet it might not be the optimal approach for routine configuration updates.

The `wasmd` Cosmos SDK module defines the migration entrypoint for the contract `admin` to migrate to a new `code_id` in addition to other state changes defined in the migration message. It is likely that there will be scenarios where the administrator wants to update a config parameter but does not want to issue a `code_id` migration. The migration entrypoint requires a `code_id` to be specified, but this id can also be the same as the current id. However, this could potentially lead to a misconfiguration if the admin intends to update a config parameter but inputs an incorrect `code_id`. Additionally, even when the same `code_id` is supplied, the wasm module will perform many unnecessary operations for a simple config update, which will result in a higher gas consumption.

Additionally, to maintain data consistency, it is crucial to verify the current version of the contract before permitting any updates to its fields. By doing so, we can confirm that the code IDs have undergone migration and their state has been accurately altered before the application of any `MigrateMsg` message. This approach ensures that updates are only applied when the version remains consistent, safeguarding against discrepancies in the contract's state.

Recommendation

We recommend moving config updates to a privileged execute entrypoint.

Status: Acknowledged

40. Redundant check for duplicates in a loop

Severity: Informational

In `contracts/remote-chain-splitter/src/contract.rs:69-80`, split configurations are iterated. A set `encountered_denoms` is used to track denominations and filter redundant values.

However, the redundant iterations do not occur because `msg.splits` is declared as value of the `BTreeMap` type, and hence only allows entries with distinct keys.

As a consequence, the duplicate check is unnecessary.

Recommendation

We recommend removing unnecessary checks for duplicates.

Status: Acknowledged

41. Inefficiencies in iterations

Severity: Informational

Unbounded iteration occurs in many locations throughout the codebase. Examples could be found in `packages/covenant-utils/src/split.rs:90` and `remote-chain-splitter/src/contract.rs:69`. Overall, we have found more than 20 occurrences of unbounded iterations.

Although advanced data structures like `BTreeSet` or `BTreeMap` are used in most cases, some iterations are suboptimal. For instance, searching for an element by key could be performed in `O(log N)` asymptotic complexity using the `find` function instead of `O(N)` traversal using `iter()`, e.g. in `packages/covenant-utils/src/split.rs:98`.

In addition, the code is not optimized for particular contexts, e.g. when the collections consist of only 2 items. Advanced tree-like data structures incur an additional memory and computation footprint in this case.

Finally, the lack of explicit limits on collections expands the attack surface by potentially allowing for DoS attacks.

Recommendation

We recommend optimizing data structures and their operations.

For example:

- If a collection consists only of 2 items, a tuple is the best choice for the container. Utility functions like `map` or `find` could be implemented trivially for this type.
- If a collection can grow arbitrarily, then a sorted vector or tree-like structure is an appropriate choice, but specialized functions like `find` should be used to reduce gas consumption.
- If a collection is mutated permissionlessly, consider capping its size.

Status: Acknowledged

42. Code duplication decreases readability and maintainability

Severity: Informational

The codebase contains multiple code duplicates:

1. Contracts `remote-chain-splitter`, `stride-liquid-staker` and `ibc-forwarder` share a lot of similarities and nearly identical code fragments, related to ICA registration and interchain token transfers.
2. Particularly, the function `get_ica` is defined 3 times, but it is still not used in multiple places where the `INTERCHAIN_ACCOUNTS` storage is queried, e.g. in `contracts/ibc-forwarder/src/contract.rs:136`. Also, the function's parameter `interchain_account_id` always has the same value of `INTERCHAIN_ACCOUNT_ID` and could be inlined.
3. The `validate` function defined in `packages/covenant-utils/src/split.rs:46-63` is used only once in the specific context of `two-party-pol-holder` contract. In this context, its effect is equivalent to the effect of the function `validate_shares` defined in `packages/covenant-utils/src/split.rs:66-80`. It is preferable to just call the latter function with prior verification that `self.receivers` value contains only the two parties.

Code duplication decreases readability and maintainability, thereby expanding the potential for security vulnerabilities.

Recommendation

We recommend refactoring the codebase to avoid code duplications. Shared modules with common code as well as generic type parameters and function-type parameters could be used to streamline the data flow.

Status: Resolved

43. Remove commented code blocks

Severity: Informational

In the contracts within the scope of this audit there are code blocks that have been commented out. It is best practice to remove all unused code before the contracts are released for better readability and maintainability of the codebase. The following instances of commented code have been found:

- `contracts/single-party-pol-holder/src/state.rs:16-22`
- `contracts/native-router/src/contract.rs:42`

Recommendation

We recommend removing all commented code blocks.

Status: Resolved

44. Remove debug messages

Severity: Informational

In the contracts in the scope of this audit, we found several debug messages and `println!` statements. It is best practice to remove all debug messages and anything that may output to stdout before the contracts are deployed to production.

Recommendation

We recommend removing all debug and print messages from the codebase.

Status: Resolved

45. Remove dead code

Severity: Informational

There are several instances of unused code in the codebase. It is best practice to remove all dead code from the codebase. Instances of dead code are:

- `ExecuteMsg` in `contracts/single-party-pol-covenant/src/msg.rs:202`
- `from_share_response` in `contracts/two-party-pol-holder/src/msg.rs:508`
- `WITHDRAW_LIQUIDITY_BALANCES_QUERY_CALLBACK_ID` defined in `contracts/osmo-liquid-pooler/src/contract.rs:46` is never actually used, and its handler in `contracts/osmo-liquid-pooler/src/polytone_handlers.rs:70-72` is never reached

Recommendation

We recommend removing all dead code from the codebase.

Status: Resolved

46. Resolve TODO code and comments

Severity: Informational

There are several instances of `todo!` macros in the codebase. It is best practice to resolve all TODOs. Instances of TODO code have been found in::

- `contracts/remote-chain-splitter/src/contract.rs:384`
- `contracts/two-party-pol-holder/src/contract.rs:653`
- `contracts/swap-holder/src/contract.rs:256`
- `contracts/two-party-pol-holder/src/contract.rs:634`

Recommendation

We recommend resolving all TODOs from the codebase.

Status: Resolved