Audit Report

Drop Contracts

v1.1

August 19, 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 Neutron Audit Sponsorship Program to perform a security audit of the Drop CosmWasm smart contracts.

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/hadronlabs-org/drop-contracts
Commit	`ba218e46fd7d8aca1f1f9a3857d01b9efcf3b1ff`

Scope	The following contracts and packages are in scope:
	
	`.`
	`├── contracts`
	`│  ├── astroport-exchange-handler`
	`│  ├── core`
	`│  ├── distribution`
	`│  ├── factory`
	`│  ├── pump`
	`│  ├── puppeteer`

	<pre>│ ├── rewards-manager
	│ ├── strategy
	│ ├── token
	│ ├── validators-set
	│ ├── withdrawal-manager
	│ └── withdrawal-voucher
	└── packages
	    ├── base
	    ├── helpers
	    ├── macros
	    └── puppeteer-base</pre>

Fixes verified at commit	`0ebf97f7715f98d1ae7fe73b1d185a7f3aed4090` 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

Drop is a liquid staking protocol designed for deployment on the Neutron chain within the Cosmos ecosystem.

The protocol leverages Inter-Blockchain Communication (IBC), Interchain Accounts (ICA) and Interchain Queries (ICQ) to facilitate seamless staking and unstaking across the Cosmos ecosystem.

Drop integrates an autocompounding feature, automatically restaking rewards to optimize yield.

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	**High**	The protocol encompasses several contracts that communicate with each other through message exchanges and callbacks, in addition to utilizing IBC messages, ICA accounts and ICQ queries.
Code readability and clarity	**Medium-High**	-
Level of documentation	**High**	The client provided very detailed documentation including diagrams of the architecture.
Test coverage	**Low**	`cargo tarpaulin` reports a `14.38%` test coverage.

Summary of Findings

No	Description	Severity	Status
1	Missing access control of strategy contract configuration updates allows attackers to DoS attack the protocol and redirect queries to malicious contracts	Critical	Resolved
2	Incorrect fee accounting leads to loss of funds and DoS	Critical	Resolved
3	The protocol can get permanently stuck in the Staking state due to unhandled errors	Critical	Acknowledged
4	First bonding results in a division by zero error	Critical	Resolved
5	Attackers can spam Unbond messages to cause DoS of the protocol	Critical	Resolved
6	Incorrect mint calculation during `DenomType::LsmShare` bonding	Critical	Externally Reported
7	Invalid FSM state transitions prevent expected flow execution	Major	Resolved
8	The refund functionality of the pump contract can be disabled with spam tokens	Major	Resolved
9	Pump contract refunds can be used to front-run push operations	Major	Resolved
10	Token contract's `UpdateOwnership` and `SetTokenMetadata` messages cannot be executed	Major	Resolved
11	The rewards manager can be disabled with spam tokens	Major	Resolved
12	Incorrect validation of IBC coin denomination	Major	Resolved
13	LSM shares are not processed until the buffer size is reached	Major	Resolved
14	Risks of validators incentivized to conduct misbehavior leading to the depegging of the liquid staking token	Major	Acknowledged
15	MEV in the exchange handler	Major	Acknowledged

16	MEV in the form of unclaimed rewards	Major	Acknowledged
17	Risk of gas exhaustion for unbounded ICA messages on the remote chain	Major	Acknowledged
18	The core contract could execute state machine handlers on outdated remote chain data	Major	Resolved
19	Extensive computation in the get_stake_msg function could cause DoS	Major	Resolved
20	No active validator is required to allow liquid staking	Major	Acknowledged
21	Staking fees could be added to an outdated fee address	Major	Resolved
22	Incorrect exchange rate tracking due to missing intermediate states	Major	Externally Reported
23	Incorrect exchange rate calculation during bonding	Major	Externally Reported
24	Batching of undelegation requests could lead to incorrect exchange rate	Major	Externally Reported
25	Potential failure of claim and undelegate operations due to limits of the `MsgSubmitTx` message	Major	Externally Reported
26	Incorrect LSM share amount tracking	Major	Externally Reported
27	Incorrect `UNBOND_BATCH_ID` calculation if the retried failed batch and the last one are not subsequent	Minor	Externally Reported
28	Arbitrary validator weights and units are configurable and the validator configuration might be overwritten	Minor	Acknowledged
29	The liquid staking denomination can be overwritten	Minor	Partially Resolved
30	IBC coin's related channel can be updated	Minor	Resolved
31	Missing validation in the pump contract	Minor	Acknowledged
32	Missing validation in the puppeteer contract configuration update	Minor	Partially Resolved
33	Any decorrelation between the liquid and underlying asset can be used by validators to hedge against slashing by shorting the liquid token	Minor	Acknowledged

34	Balance and delegations might not be queried for a new validator	Minor	Resolved
35	Unbounded iterations over the validator set might run out of gas	Minor	Acknowledged
36	Lack of grouped versioning of code IDs could lead to malfunctioning deployments	Minor	Acknowledged
37	Missing validation during the factory contract instantiation	Minor	Resolved
38	The strategy contract ownership cannot be transferred	Minor	Resolved
39	The factory contract allows the owner to execute arbitrary messages	Minor	Acknowledged
40	Missing funds in the token contract required to pay the `DenomCreationFee` prevents the protocol from being deployed	Minor	Acknowledged
41	The two-step ownership transfer can be bypassed	Minor	Partially Resolved
42	Misconfiguration of `unbonding_safe_period` could lead to DoS the FSM	Minor	Acknowledged
43	Reward handlers can be overwritten	Minor	Resolved
44	Missing validation during the validator-set contract configuration update	Minor	Resolved
45	Missing validation during the validator-set contract validators' info update	Minor	Resolved
46	Missing validation during the core contract instantiation	Minor	Resolved
47	IBC denom validation does not support multi-hop routed tokens	Minor	Acknowledged
48	The `BONDED_AMOUNT` is not decreased during unbondings leading to incorrect limit validation	Minor	Partially Resolved
49	Missing validation during the withdrawal-manager contract instantiation	Minor	Resolved
50	Missing validation during core contract configurations update	Minor	Resolved
51	Neutron governance updating IBC fees could temporarily hinder interchain transactions	Minor	Resolved

52	Missing validation during core contract configurations update	Minor	Partially Resolved
53	Local height tracking could lead to incorrect ICQ results	Minor	Externally Reported
54	Unhandled overflow	Minor	Externally Reported
55	`LAST_PUPPETEER_RESPONSE` is not being reset on `Undelegate` error	Informational	Resolved
56	The `IBCTransfer` callback handler is not being used	Informational	Resolved
57	Callback execution without access control	Informational	Resolved
58	Missing use of checked math functions to prevent overflow	Informational	Acknowledged
59	“Migrate only if newer” pattern is not followed	Informational	Resolved
60	Incorrect event emissions	Informational	Resolved
61	Resolve TODOs	Informational	Acknowledged
62	Misleading function name and documentation	Informational	Resolved
63	Miscellaneous Comments	Informational	Partially Resolved

Detailed Findings

1. Missing access control of strategy contract configuration updates allows attackers to DoS attack the protocol and redirect queries to malicious contracts

Severity: Critical

In `contracts/strategy/src/contract.rs:187-226`, the `exec_config_update` function of the strategy contract allows updating the contract configurations such as the core, puppeteer, validator-set and distribution contract addresses, as well as the reference `denom`.

However, since there is no access control for the function, anyone can update the contract's config to arbitrary values.

Consequently, attackers can consistently spam `UpdateConfig` messages to store invalid addresses for contract dependencies causing the strategy contract's queries to fail and disrupting protocol operations.

Additionally, attackers could update contract dependencies to malicious contracts that return malicious data.

Recommendation

We recommend implementing access control to only allow the owner to update the contract configuration.

Status: Resolved

2. Incorrect fee accounting leads to loss of funds and DoS

Severity: Critical

The `get_stake_msg` function, defined in `contracts/core/src/contract.rs:1063-1127`, processes delegation messages, updating the `COLLECTED_FEES` with the amount to be sent to the respective `fee_address`.

Those fees are then transferred from the ICA account to the fee collectors during the execution of the `get_non_native_rewards_and_fee_transfer_msg` function defined in lines `1301-1314`.

However, after the fees are transferred by the puppeteer contract to fee collectors, the `COLLECTED_FEES` map is not reset.

Consequently, as the FSM continues normal operation, more fees are added and all the fees, including the ones that have been already sent, are transferred again. This leads to a loss of funds for the users and eventually causes a denial of service, due to the impossibility of transferring an amount larger than the balance of the ICA account.

Recommendation

We recommend resetting `COLLECTED_FEES` after fees are transferred to the fee collectors.

Status: Resolved

3. The protocol can get permanently stuck in the `Staking` state due to unhandled errors

Severity: Critical

In `contracts/core/src/contract.rs:693-696`, the `execute_tick_staking` function within the core contract acts as the designated handler when the `ContractState` transitions to `Staking` during routine procedures.

However, if during this process the `puppeteer` contract callback to the `PuppeteerHook` stores a `ResponseHookMsg::Error` because of a timeout or an error in the ICA transaction, the `execute_tick_staking` function consistently returns an error in line `695`.

Consequently, lacking an alternative handler to progress the `ContractState`, the routine becomes unable to proceed, resulting in the protocol becoming permanently stuck.

Recommendation

We recommend that after a desired number of staking retries, the protocol enters an emergency state e.g. by pausing the whole state machine including all components, and securing users' funds.

Status: Acknowledged

The client acknowledges this issue and aims to address it in a future revision.

4. First bonding results in a division by zero error

Severity: Critical

In `contracts/core/src/contract.rs:173-179`, the `query_exchange_rate` function calculates the actual `exchange_rate` with the following formula:

$$exchange\_rate = \frac{delegations\_amounts + core\_balance + tota\_lsm\_shares - current\_stake - unprocessed\_unbonded\_amount}{ld\_total\_amount}$$

However, during the first bonding, it results in a zero `exchange_rate` since `delegations_amount`, `total_lsm_shares` and `unprocessed_unbonded_amount` are zero while `core_balance` and `current_stake` would be the same amount.

Consequently, this would result in a division by zero error in `contracts/core/src/contract.rs:788`, causing the protocol to not be able to process bonding.

Recommendation

We recommend revising the calculation of the `exchange_rate` for the first bond.

Status: Resolved

5. Attackers can spam `Unbond` messages to cause DoS of the protocol

Severity: Critical

In `contracts/core/src/contract.rs:952-959`, the `execute_unbond` function of the core contract allows users to unbond their tokens.

By doing so, the execution appends an `UnbondItem` to the `unbond_items` vector of the current `unbond_batch` and then stores it in the contract.

However, since there is no maximum cardinality of items that can be appended in the `unbond_items` vector and no minimum unbounding amount requirement, attackers could spam a large amount of `Unbond` messages with a small unbounding amount to grow the vector size.

This would result in increasing the costs for encoding and storing the `unbond_items` vector to the point of the execution running out of gas and hindering users to unbond.

Recommendation

We recommend using a map instead of a vector to store `UnbondItem` entries.

Status: Resolved

6. Incorrect mint calculation during `DenomType::LsmShare` bonding

Severity: Critical

In `contracts/core/src/contract.rs:1005-1064`, the `execute_bond` function enables users to bond their tokens and mint derivatives.

However, if the `denom_type` is equal to `DenomType::LsmShare`, the amount is directly used without calculating the underlying amount.

Consequently, an incorrect number of tokens is computed and minted, resulting in inaccurate accounting and potential financial losses for users.

This issue has been independently reported by a third party.

Status: Externally Reported

7. Invalid finite state machine transitions prevent expected flow execution

Severity: Major

There are multiple instances of state transitions that are not considered valid in the context of the finite state machine (FSM).

For example, in the function `execute_tick_idle`, in `contracts/core/src/contract.rs:562`, the FSM state is set to `Claiming`, even if there are no `validators_to_claim`. Since the state transition from `Claiming` to `Unbonding` is not valid this would prevent the branch from being executed in lines `574-579`.

However, even if the state of the FSM remained in the `Idle` state, the only valid transition to `Unbonding` is from `Staking`. This would prevent any unbondings in any other tick except `execute_tick_staking`.

In particular, in `execute_tick_claiming` the branch cannot be executed in lines `651-654`. Similarly, in `execute_tick_transferring` the branch in lines `676-679` would be an invalid state transition from `Transferring` to `Unbonding`.

Recommendation

We recommend updating the FSM state to `Claiming` only if there are rewards to be claimed and adding the necessary state transitions in the valid set in `packages/base/src/state/core.rs:109-153`.

Status: Resolved

8. The refund functionality of the pump contract can be disabled with spam tokens

Severity: Major

In `contracts/pump/src/contract.rs:105-110` the `execute_refund` function allows sending all the coins held by the pump contract to the `refundee` address.

However, since the contract iterates and sends all the coins it holds, attackers can cause a denial of service by creating spam tokens and sending them to the contract. This attack is particularly cheap on Neutron as there is no fee for creating tokens in the `tokenfactory` module.

Consequently, the refund functionality of the pump contract would be disabled.

Recommendation

We recommend allowing the function caller to specify a set of denoms to be refunded.

Status: Resolved

9. Pump contract refunds can be used to front-run push operations

Severity: Major

In `contracts/pump/src/contract.rs:105-110` the `execute_refund` function allows sending all the coins the pump contract holds to the `refundee` address.

However, since the pump contract is permissionless, refunds can be consistently executed by attackers to prevent the balance of the contract from being pushed to the `dest_address`, frontrunning push operations.

Recommendation

We recommend allowing only the owner to execute refunds or remove the functionality completely if not necessary.

Status: Resolved

10. Token contract's `UpdateOwnership` and `SetTokenMetadata` messages cannot be executed

Severity: Major

In `contracts/token/src/contract.rs:53-77` the token contract defines messages that can be executed by the core contract, including `Mint`, `Burn`, `UpdateOwnership` and `SetTokenMetadata`.

However, since the core contract does not define any calls to `SetTokenMetadata` or `UpdateOwnership`, the two functions cannot be executed.

Consequently, any attempts by the owner to update the ownership or set the token metadata would fail because of the access control.

Recommendation

We recommend ensuring that `UpdateOwnership` and `SetTokenMetadata` can be executed by the owner, instead of the core contract.

Status: Resolved

11. The rewards manager can be disabled with spam tokens

Severity: Major

In `contracts/rewards-manager/src/contract.rs:171-176` an unbounded iteration over all token balances can run out of gas and disable the `exec_exchange_rewards` function.

However, since the contract iterates and sends all the coins it holds, attackers can cause a denial of service by creating spam tokens and sending them to the contract. This attack is particularly cheap on Neutron as there is no fee for creating tokens in the `tokenfactory` module.

Recommendation

We recommend limiting the iteration by implementing logic that allows specific token swapping.

Status: Resolved

12. Incorrect validation of IBC coin denomination

Severity: Major

In `contracts/core/src/contract.rs:1458` the function `check_denom` checks whether either the port is `transfer` or the channel is `config.channel` to ensure that the correct denomination is sent to the protocol.

However, this would allow the processing of coins from the correct channel but with a port different from the ibc-transfer module's `transfer` or a coin from the correct port but from an unknown channel.

We are reporting this issue as major since the attack vectors are limited. However, this could allow the use of coins from a port different from `transfer` or from a malicious channel.

Recommendation

We recommend requiring both the channel and the port parts of the coin denom to be correct. Additionally, we recommend resolving and implementing the TODO comment in line `1444`.

Status: Resolved

13. LSM shares are not processed until the buffer size is reached

Severity: Major

In `contracts/core/src/contract.rs:1331` the function `get_pending_redeem_msg` is used to construct the LSM shares redemption message.

In lines `1342-1347`, the function checks that there are enough pending shares to be processed and that the `lsm_redeem_maximum_interval` is reached.

However, in case of an insufficient amount of shares, they will not be processed, irrespective of the maximum timeout set leading to delays and missing rewards for LSM stakers during low-volume periods.

Recommendation

We recommend proceeding with the redemption process if the `lsm_redeem_maximum_interval` elapsed, even if the threshold has not been met.

Status: Resolved

14. Risks of validators incentivized to conduct misbehavior leading to the depegging of the liquid staking token

Severity: Major

The form of liquid staking implemented introduces new incentives that require some form of incentive risk management.

In this report, we have identified three minor issues that relate to this issue:

- No active validator is required to allow liquid staking
- Arbitrary validator weights and units are configurable and the validator configuration might be overwritten
- Any decorrelation between the liquid and underlying asset can be used by validators to hedge against slashing by shorting the liquid token

While these issues individually are classified as minor, they can influence each other and indicate that a general risk management framework should be developed to align incentives

under adverse conditions, such as possible collusion of validators, inactivity, and depegging due to non-overlapping validator sets.

Usually, we assume aligned validators. However, we have to relax this assumption in the case of a liquid staking protocol that deals with a whitelisted set of validators.

Generally, validators are aligned through a combination of financial incentives (mainly staking rewards and slashing). Their responsibility is to guarantee both network uptime and non-malicious behavior. The slashing mechanism ensures that they act in the best interest of the network, maintaining its security and reliability.

With the introduction of a liquid staking token run on a network of whitelisted validators, an additional layer or game theory is introduced. For illustrative purposes, it is worth studying edge cases where there is only one whitelisted validator on Drop that, at the same time, has the smallest weight in the active 180 validators on the underlying chain. In that case, malicious actions have negligible influence on the underlying asset or the consensus, but the value of the liquid staking token will be completely exposed to their misbehavior. In such a situation, a validator might face the incentive to get slashed while shorting the liquid staking token. This remains true if only two validators are in the liquid staking whitelist but collude.

Another edge case would be a 34% collusion of validators on the main chain holding the majority of weight in the liquid staking protocol. They would still be able to hedge against slashing because the liquid staking token would be overly exposed to their behavior and additionally be able to block the consensus on the main chain.

These situations can be generalized further:

Depegging is more likely if there is less overlap between the validator sets of the liquid staking and the base asset. Stronger depegging tendencies (decorrelation) foster the ability of validators to hedge against slashing. Overlap of the validator set is therefore desirable.

The more weight one validator has in the network of active whitelisted validators, the higher the liquid token price can be correlated to the misbehavior of that validator. A larger weight can be caused by both a small set of active validators and large pre-configured weights of single validators.

Hence, a strong diversification is desirable. This can, for example, be achieved by relatively equal but small individual (pre-configured and effective) weights of validators.

Recommendation

We recommend implementing short-term and long-term measures. In the short term, fixing all the minor issues can be a remedy. In the long term, an automated (oracle-like) on-chain whitelisting procedure for validators that maximizes the overlap of the validator sets of the underlying and the liquid asset while maintaining diversification is recommended.

Status: Acknowledged

The client states that the issue is relevant to any protocol that operates with a permissioned list of validators. They note that the issue becomes significant only when the penetration of a

liquid staking protocol is substantial. With low penetration, the protocol does not create sufficient economic incentives for a validator to misbehave. From a product development perspective, prioritizing automated set management in the early stages is sub-optimal. Implementing it early will not significantly contribute to the protocol's success and may dilute the focus. However, as the protocol grows, automated set management becomes increasingly crucial, so it should be developed in parallel.

For the long term, the client mentions that they already have a proof of concept for automated validator management. This includes gathering information about validators' metrics with ICQ and allowing the addition of more on-chain inputs. In the short term, the issue will be primarily managed through reasonable validator set management.

15. MEV in the exchange handler

Severity: Major

In `contracts/astroport-exchange-handler/src/contract.rs:207-235`, the permissionless function `exec_exchange` automatically trades rewards on Astroport without setting a belief price or maximum spread.

While there is a default maximum spread on Astroport of 0.5%, this does not ensure that the maximal extractable value is limited to this number.

An attacker or MEV searcher can create an atomic transaction that calls the function after manipulating or influencing the asset's price in the pool to execute arbitrage around the price point. Even if the function was not permissionless a frontrunner or validator could extract value by sandwiching the transaction.

As a consequence the state machine can be partially disabled, preventing the distribution of rewards.

Recommendation

We recommend implementing a belief price and maximal spread.

Status: Acknowledged

16. MEV in the form of unclaimed rewards

Severity: Major

The exchange rate computations in `contracts/core/src/contract.rs:173-180` do not consider unclaimed rewards.

Consequently, users could receive more ldTOKENs than they should when staking to the protocol. MEV searchers, validators and frontrunners can exploit this when there are sufficient pending rewards to receive additional tokens by creating a staking transaction right before

the claiming message. They could also trigger the claiming message themselves with the `ExecuteMsg::Tick`.

Recommendation

We recommend, in the short term, implementing the measures described in the documentation such as approximating future rewards.

Long-term measures should include querying rewards and updating the state when the exchange rate is calculated, but could also require a redesign of the state machine as eventually, it might be necessary to claim rewards before each bonding and unbonding, which is not in line with the current design of the state machine.

Status: Acknowledged

The client acknowledges that if a user stakes assets right before the claim-stake FSM process, they receive more tokens due to the presence of more unclaimed rewards. However, the only way to benefit from this would be immediate unstaking, which, despite providing an instant upside, exposes the funds to the risk of slashing and yields no rewards during the undelegation process. Thus, the risk profile remains the same as staking, but with lower rewards since only one period's rewards are received.

The client emphasizes their intention to mitigate the potential impact of unclaimed rewards not being accounted for. They consider reward estimation to be risky because, while it can reduce the difference between the real (including unclaimed rewards) and assumed (excluding unclaimed rewards) exchange rates, it can also lead to situations where the estimated exchange rate exceeds the real one, causing an undesirable drop in the exchange rate.

As a long-term solution, the client suggests either increasing the frequency of reward processing, potentially by leveraging the authz module or adjusting the state machine, or querying unclaimed rewards, which would require additional research. Both options, however, would increase protocol operation costs. The final solution remains to be determined for future versions.

17. Risk of gas exhaustion for unbounded ICA messages on the remote chain

Severity: Major

In `contracts/puppeteer/src/contract.rs:524` and `673`, the `compose_submsg` function is leveraged to submit ICA transactions. The function takes a slice of message elements intended for execution on the host chain, serializes them, and dispatches an ICA transaction.

However, there is no upper bound defined for the cardinality of messages that can be dispatched. There is a potential risk of the relayer encountering an out-of-gas error in the host chain due to the significant amount of messages.

This scenario is plausible in messages like `Delegate` and `Undelegate` which generate a message for each delegation or undelegation item and bundles them into a single ICA transaction.

Consequently, if the execution of messages on the remote chain runs out of gas, the core contract would not be able to stake or unstake coins anymore.

Recommendation

We recommend introducing a batching mechanism to split messages to the host chain in multiple transactions.

Status: Acknowledged

The client states that they are currently addressing this issue by using a limited validator set size.

18. The core contract could execute state machine handlers on outdated remote chain data

Severity: Major

In `contracts/puppeteer/src/contract.rs:904-911`, the `SudoMsg::KVQueryResult` handler within the puppeteer contract manages replies from interchain queries and archives their outcomes within the contract for subsequent retrieval by the core contract.

According to the documentation, these queries undergo execution by an off-chain coordinator before the core contract's `tick` method execution.

However, the protocol does not enforce this in the code, lacking a mechanism for the core contract to reject stale data.

As a consequence, in the event of malfunctioning of the off-chain orchestrator, the contract might execute operations based on outdated data leading to incorrect operations and state transitions.

Recommendation

We recommend tagging retrieved data with timestamps in the puppeteer contract and rejecting outdated data in the core contract.

Status: Resolved

19. Extensive computation in the `get_stake_msg` function could cause DoS

Severity: Major

In `contracts/core/src/contract.rs:1063-1127`, the `get_stake_msg` of the core contract executes multiple messages and smart queries to calculate the `IdealDelegations` to submit to the remote chain:

- A `QueryMsg::CalcDeposit` query of the strategy contract is performed, iterating all the actual delegations and the validator set.
- A `QueryMsg::CalcDeposit` query of the `Distribution` contract is performed, iterating the delegations provided by the strategy contract three times respectively in `calculate_total_stake_deposit`, `calc_ideal_stake`, and `distribute_stake_deposit` functions.
- Retrieved `IdealDelegations` are iterated and encoded in the puppeteer contract to serialize them for executing them on the remote chain.

Consequently, if the cardinality of validators and delegations is substantial, the delegation process could run out of gas leading the protocol to be unable to perform state machine transitions.

Recommendation

We recommend optimizing the aforementioned functions to reduce the number of iterations as well as implementing a maximum number of whitelisted validators.

Status: Resolved

20. No active validator is required to allow liquid staking

Severity: Major

Before the delegation of stake, there is no logic implemented that checks the activity of a validator.

Therefore funds might be delegated to offline validators, putting the funds of users at the foreseeable risk of being slashed.

In the worst case scenario, all validators that receive delegations could be offline. This could be the result of a small configured validator set, an eclipse attack, validator misbehavior or collusion.

Recommendation

We recommend requiring a minimum number of active validators and a higher minimum weight threshold (e.g., 33% of the listed total weight) to enable users to perform liquid staking. We also recommend only staking to online validators. This would also drastically reduce the

rebalancing needs if validators reactivate. Alternatively, the users' funds could be parked on a custodial account until a minimum number of validators and weights are active.

Status: Acknowledged

The client states that querying the status of validators for adding them to the set makes sense and should be done on an ongoing basis. While a validator may be active when added, they can drop out of the active set later, so continuous checking is necessary but not entirely sufficient. Ongoing validator set management is complex, requiring both low-level mechanics for querying this information and high-level business logic for handling it properly. They are actively working on this and plan to include it in future versions.

21. Staking fees could be added to an outdated fee address

Severity: Major

In `contracts/src/contract.rs:1063-1127`, the `get_stake_msg` function constructs the delegation messages and calculates staking fees, updating `COLLECTED_FEES` accordingly.

However, if there is some `fee_item`, the new fee amount is added to the previous item, irrespective of the `fee_address`.

Consequently, any changes to the `config.fee_address` would have no impact as new fees would continue to be added to the old fee address.

Recommendation

We recommend ensuring that new fees are deposited to the actual `config.fee_address`.

Status: Resolved

22. Incorrect exchange rate tracking due to missing intermediate states

Severity: Major

In `contracts/core/src/contract.rs:502-530`, the `execute_tick_idle` function is responsible for checking whether the minimum idle interval has elapsed. If the interval has not passed, the function sends messages constructed using functions such as `get_non_native_rewards_and_fee_transfer_msg`, `get_pending_redeem_msg`, or `get_pending_lsm_share_msg`.

However, the state is not transitioned from the `Idle` state, which can lead to issues with calculating the correct exchange rate.

This issue has been independently reported by a third party.

Status: Externally Reported

23. Incorrect exchange rate calculation during bonding

Severity: Major

The `execute_bond` function, located in `contracts/core/src/contract.rs:752-806`, enables users to bond their tokens and mint derivatives.

However, some values, such as `TOTAL_LSM_SHARES`, are updated prior to fetching the exchange rate using the `query_exchange_rate` function. This sequence results in an incorrect exchange rate calculation, as it includes the current bonding transaction.

Consequently, users bonding `DenomType::LsmShare` receive a smaller amount of minted tokens than they should.

This issue has been independently reported by a third party.

Status: Externally Reported

24. Batching of undelegation requests could lead to incorrect exchange rate

Severity: Major

Performing assets in undelegation batches can lead to issues.

Particularly, if nearly all assets are undelegated and slashing occurs, it may result in a negative exchange rate since undelegated assets are subtracted from the numerator of the formula.

This issue has been independently reported by a third party.

Status: Externally Reported

25. Potential failure of claim and undelegate operations due to limits of the `MsgSubmitTx` message

Severity: Major

The `execute_claim_rewards_and_optionally_transfer` and `execute_undelegate` functions, defined in `contracts/puppeteer/src/contract.rs:653-706` and `708-749`, construct and

send a `MsgSubmitTx` containing multiple `MsgWithdrawDelegatorReward` and `MsgUndelegate` messages.

However, there is a limit to the number of inner messages allowed in `MsgSubmitTx`, specified by the `msg_submit_tx_max_messages` parameter.

Consequently, this constraint could cause large batches of messages to fail.

This issue has been independently reported by a third party.

Status: Externally Reported

26. Incorrect LSM share amount tracking

Severity: Major

In `contracts/core/src/contract.rs`, when performing computations on LSM shares, there is an inconsistency in tracking the share amount.

Specifically, during certain operations like `execute_bond`, the share number is used to represent the amount of underlying ATOM delegated. However, in other functions, such as `execute_puppeteer_hook`, the same share number is treated as the number of shares.

This discrepancy could result in fluctuations in exchange rates, unaccounted-for shares and failures in processing LSM tokens.

This issue has been independently reported by a third party.

Status: Externally Reported

27. Incorrect `UNBOND_BATCH_ID` calculation if the retried failed batch and the last one are not subsequent

Severity: Minor

In `contracts/core/src/contract.rs:1129-1183`, the `get_unbonding_msg` function is responsible for generating `Undelegate` messages for the current batch or retrying a previously failed batch.

However, when handling a failed batch, the function erroneously updates the `UNBOND_BATCH_ID` by incrementing the `executed_batch_id` by one.

Consequently, if the failed batch and the last executed batch are not consecutive, this could result in the unintended re-execution of batches.

This issue has been independently reported by a third party.

Status: Externally Reported

28. Arbitrary validator weights and units are configurable and the validator configuration might be overwritten

Severity: Minor

The `execute_update_validator` function in `contracts/validators-set/src/contract.rs:155-170` does not validate the weight of a validator to be in a certain unit or range. It allows for arbitrary weights in arbitrary magnitudes. This is error-prone and a deviation from best practices. Additionally, as highlighted in the issue Risk that validators are incentivized to conduct misbehavior that leads to depegging of the liquid staking token, a rule-based logic and similar weights for the validator set are desirable.

In addition, the `validator.valoper_address` is not validated and the validator configuration might be overwritten as there is no check if a validator already exists.

Recommendation

We recommend implementing logic that requires the weights to be within a certain range and the same order of magnitude. Additionally, we recommend a rule-based allocation of weights such as 1/N (as a conservative option), a power law (a less conservative option), or choosing the weights based on historical weights as an off-chain option.

Status: Acknowledged

The client states that, at the current stage, they want to maintain control over the validator set and distribution to manually handle various situations, such as strikes for validators due to misbehavior. They plan to add more automated rules for validator set management in future versions.

29. The liquid staking denomination can be overwritten

Severity: Minor

In

- `contracts/core/src/contract.rs:915-917`,
- `contracts/astroport-exchange-handler/src/contract.rs:170`,
- `contracts/pump/src/contract.rs:150`, and
- `contracts/puppeteer/src/contract.rs:262`,

the liquid staking denomination `ld_denom` can be reset by the `execute_update_config` function. Resetting the denomination while the protocol is operational disables all unbonding and most other protocol operations.

Recommendation

We recommend prohibiting modification of the `ld_denom` once it is set.

Status: Partially Resolved

30. IBC coin's related channel can be updated

Severity: Minor

In `contracts/core/src/contract.rs:923-925` the configured `config.channel` can be updated by the `execute_update_config` function.

However, since this channel is used to identify IBC coins, this might lead to misconfiguration and allow the minting of liquid staking tokens from invalid coins.

Recommendation

We recommend allowing IBC channel updates only in migrate handlers for emergency purposes.

Status: Resolved

31. Missing validation in the pump contract

Severity: Minor

In `contracts/pump/src/contract.rs:45-62` and `contracts/pump/src/contract.rs:126-152`, during the execution of the `instantiate` and `execute_update_config` functions of the pump contract, data provided by the owner is stored in the contract without any validation.

Recommendation

We recommend validating input data before storing it in the contract.

Status: Acknowledged

32. Missing validation in the puppeteer contract configuration update

Severity: Minor

In `contracts/puppeteer/src/contract.rs:240-291`, during the execution of the `execute_update_config` function of the puppeteer contract, data provided by the owner is stored in the contract without any validation.

However, it should be validated to not allow for incorrect configurations.

Recommendation

We recommend validating input data before storing it in the contract.

Status: Partially Resolved

33. Any decorrelation between the liquid and underlying asset can be used by validators to hedge against slashing by shorting the liquid token

Severity: Minor

As the liquid staking token is tightly correlated to the behavior of the whitelisted validators, they might use a short position in the liquid token to hedge against slashing. If they are slashed, the price of the liquid token will fall.

Consequently, they can leverage a short position in the liquid token to mitigate or offset the slashing effects, introducing undesired volatility of the liquid staking derivative.

Recommendation

We recommend a strongly diversified validator set so that single validator behavior can only marginally influence the prices. Additionally, we recommend disclosure agreements with whitelisted validators regarding their asset positions and legal agreements that forbid them from opening any short positions in the liquid staking token. As a long-term solution, we recommend that all validators be whitelisted only if they comply with on-chain logic.

Status: Acknowledged

34. Balance and delegations might not be queried for a new validator

Severity: Minor

In `contracts/factory/src/contract.rs:240-305`, the function `execute_proxy_msg` allows the owner to execute proxy messages on the contracts, including updating the validators of the validator set contract.

After calling `UpdateValidators`, the puppeteer contract message `RegisterBalanceAndDelegatorDelegationsQuery` is executed to register a query for retrieving balances and delegations. However, if `UpdateValidator` is called instead, no query is performed.

Consequently, any validators added using `UpdateValidator` will have their balances and delegations neither queried nor stored, and these validators will not be utilized by the protocol.

Recommendation

We recommend calling `RegisterBalanceAndDelegatorDelegationsQuery` after `UpdateValidator`, passing `info.funds` to the query instead of the `UpdateValidator` message.

Status: Resolved

35. Unbounded iterations over the validator set might run out of gas

Severity: Minor

In `contracts/validators-set/src/contract.rs:302-322`, the execution of the `execute_update_validators_voting` function might run out of gas if the `VALIDATORS_SET` becomes too large.

We are reporting this issue with minor severity since it can only be caused by the owner.

Recommendation

We recommend defining a maximum length for the `VALIDATORS_SET` vector.

Status: Acknowledged

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

Severity: Minor

The factory contract, in `contracts/factory/src/contract.rs:65`, enables instantiators to provide a specific code ID for 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 protocol.

Additionally, it poses a security risk, as users could supply incorrect or harmful code IDs.

Recommendation

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

Status: Acknowledged

37. Missing validation during the factory contract instantiation

Severity: Minor

In `contracts/factory/src/contract.rs:45-72`, during the execution of the `instantiate` function of the factory contract, data provided by the instantiator in the `InstantiateMsg` is stored in the contract without any validation.

However, data such as `remote_opts` or `token_metadata` should be validated to ensure they are correctly constructed. It should also be validated that the `salt`, `sdk_version` and `subdenom` are non-empty strings.

Recommendation

We recommend validating input data before storing it in the contract.

Status: Resolved

38. The strategy contract ownership cannot be transferred

Severity: Minor

The strategy contract does not define any methods to transfer ownership to a new address, unlike all the other contracts.

This could potentially lead to loss of control of the contract if the owner account is lost or compromised.

Recommendation

We recommend defining a method to update the strategy contract's ownership.

Status: Resolved

39. The factory contract allows the owner to execute arbitrary messages

Severity: Minor

In `contracts/factory/src/contract.rs:194-201`, the `execute_admin_execute` function within the factory contract permits the owner to execute arbitrary messages.

This practice is discouraged due to the potential for misuse, as it allows any message to be sent on behalf of the contract.

Recommendation

We recommend removing the `execute_admin_execute` function, and implementing all the paths that require owner execution through the factory contract.

In case of emergency scenarios, the migrate handler could be used instead.

Status: Acknowledged

40. Missing funds in the token contract required to pay the `DenomCreationFee` prevents the protocol from being deployed

Severity: Minor

The `instantiate` function of the token contract, defined in `contracts/token/src/contract.rs:24-50`, sends a `NeutronMsg::CreateDenom` message to the `tokenfactory` module to create a new coin with the provided `subdenom`.

However, since the creation of a new coin requires the payment of a `DenomCreationFee` and the `Init` transaction of the `factory` contract does not forward funds to this contract during the instantiation, in case the `DenomCreationFee` is set, the protocol deployment would revert.

We are reporting this issue with minor severity, since at the time of writing this report, the `DenomCreationFee` is not set in Neutron's `tokenfactory` module.

Recommendation

We recommend redirecting funds to the `token` contract to support the payment of the `DenomCreationFee` during the execution of the `Init` transaction of the `factory` contract.

Status: Acknowledged

41. The two-step ownership transfer can be bypassed

Severity: Minor

The `execute_update_config` function allows the `owner` to update the contract configurations.

However, in

- `contracts/validators-set/src/contract.rs:119-126`,
- `contracts/withdrawal-manager/src/contract.rs:145`,
- `contracts/rewards-manager/src/contract.rs:125`,
- `packages/puppeteer-base/src/execute.rs:58`,
- `contracts/puppeteer/src/contract.rs:251`,
- `contracts/pump/src/contract.rs:142` (here with the inconsistent variable name `admin`), and
- `contracts/astroport-exchange-handler/src/contract.rs:141`,

it allows updating the `owner` address bypassing the two-step ownership transfer mechanism implemented with the `UpdateOwnership` transaction.

Recommendation

We recommend not allowing owner updates during configuration updates.

Status: Partially Resolved

42. Misconfiguration of `unbonding_safe_period` could lead to DoS of the FSM

Severity: Minor

In `contracts/core/src/contract.rs:435-440`, the `execute_tick_idle` function ensures that there is no unbonding batch that will happen soon, otherwise, it does not allow the FSM to proceed with the normal execution flow to prevent mixing withdrawals with deposits.

However, If the `unbonding_safe_period` is too large, consecutive unbondings could prevent the FSM from normal execution processing for a prolonged time.

Recommendation

We recommend defining limits for the `unbonding_safe_period`.

Status: Acknowledged

43. Reward handlers can be overwritten

Severity: Minor

In `contracts/reward-manager/src/contract.rs:132-150`, the `exec_add_handler` function allows the owner to add a handler for each rewards denom.

However, the function does not check whether the denom already exists, which might lead to accidentally overwriting a current reward handler.

Recommendation

We recommend ensuring that the denom handler does not already exist.

Status: Resolved

44. Missing validation during the validator-set contract configuration update

Severity: Minor

The `execute_update_config` of the validators-set contract allows the `owner` to update the contract configuration.

However, in `contracts/validators-set/src/contract.rs:128-139`, it allows the owner to update `stats_contract` and `provider_proposals_contract` addresses without validating them.

As a consequence, incorrect addresses could be stored in the contract, leading to the impossibility of the validators-set contract to execute messages or queries required for its operations.

Recommendation

We recommend validating `stats_contract` and `provider_proposals_contract` addresses before storing them in the contract.

Status: Resolved

45. Missing validation during the validator-set contract validators' info update

Severity: Minor

The `execute_update_validators_info` function of the validators-set contract, defined in `contracts/validators-set/src/contract.rs:226-277`, allows the `stats_contract` to update the validators' information.

However, the provided update is not validated to be consistent with the existing information.

As a consequence, it could be possible to store a `last_processed_local_height` or `last_processed_remote_height` smaller than the previous stored value.

Recommendation

We recommend implementing data validation checks in the `execute_update_validators_info` function to ensure consistency with existing stored information.

Status: Resolved

46. Missing validation during the core contract instantiation

Severity: Minor

In `contracts/core/src/contract.rs:39-54`, during the execution of the `instantiate` function of the core contract, the data provided by the factory in the `InstantiateMsg` is stored in the contract without any validation.

As a consequence, it could be possible to store invalid addresses or zero durations.

Recommendation

We recommend validating input data provided in the `InstantiateMsg` before storing it in the contract.

Status: Resolved

47. IBC denom validation does not support multi-hop routed tokens

Severity: Minor

The `check_denom` function of the core contract, defined in `contracts/core/src/contract.rs-1445-1460`, checks the denom of the coin provided form the sender to define its type.

However, when checking an IBC denom, it queries the `DenomTrace` from the `ibc-go transfer` module and then splits the retrieved `path` by the `/` symbol into the `port` and `channel` strings without checking if multiple `/` symbols are present.

If the denom is not native to the remote chain, the `path` could include more than one `port/channel` pair, leading to inaccurate retrieved information and preventing the correct recognition of the `channel`.

Recommendation

We recommend parsing the `path` of the retrieved IBC denom iteratively to retrieve the correct `port` and `channel`.

Status: Acknowledged

48. The `BONDED_AMOUNT` is not decreased during unbondings leading to incorrect limit validation

Severity: Minor

The core contract keeps track of the bonded amount to ensure that the bondings have not reached the `bond_limit`.

The `BONDED_AMOUNT` is increased in the `execute_bond` function in `contracts/core/src/contract.rs:765`, however, the tracker is not updated correspondingly in the `execute_unbond` function.

Also, the `execute_reset_bonded_amount` function in `contracts/core/src/contract.rs:256` is used to reset the amount to zero, however, the `bond_limit` should be used to control the bonded funds and can be increased by updating the config.

Consequently, this could erroneously prevent users from bonding, even if the actual bonded amount is less than the limit.

We are reporting this issue with minor severity since the owner is able to reset the `BONDED_AMOUNT`.

Recommendation

We recommend decreasing the `BONDED_AMOUNT` during unbondings, and removing the `execute_reset_bonded_amount` function.

Status: Partially Resolved

49. Missing validation during the withdrawal-manager contract instantiation

Severity: Minor

In `contracts/withdrawal-manager/src/contract.rs:28-55`, during the execution of the withdrawal-manager contract `instantiate` function, input data provided in the `InstantiateMsg` is stored directly in the contract without undergoing any validation.

As a consequence, it could be possible to store invalid addresses in the `core_contract` and `withdrawal_voucher_contract` as well as an empty `base_denom`.

Recommendation

We recommend validating input data provided in the `InstantiateMsg` before storing it in the contract.

Status: Resolved

50. Missing validation during core contract configurations update

Severity: Minor

In `contracts/core/src/contract.rs:936-943`, the `execute_update_config` of the core contract allows the factory contract to update its configurations.

However, the new data is stored in the `CONFIG Item` without performing any validation.

This allows the storage of incorrect data which could disrupt the protocol's liveliness.

Recommendation

We recommend validating new configurations provided to the `execute_update_config` function before storing them in the contract

Status: Resolved

51. Neutron governance updating IBC fees could temporarily hinder interchain transactions

Severity: Minor

The contracts in scope construct the `fee` object required by Neutron's `feeRefunder` and `interchaintxs` Cosmos SDK modules with fees defined by the owner.

However, since Neutron's governance can update the required IBC fees, static 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

52. Missing validation during core contract configurations update

Severity: Minor

In `contracts/core/src/contract.rs:270-284`, the `execute_set_non_native_rewards_receivers` function of the core contract allows the owner to store multiple `NonNativeRewardsItem` in the contract

However, the provided data is stored without performing any validation. Specifically, addresses such as `address` and `fee_address` should be validated, along with ensuring that the fee falls within the `[0,1)` range.

As a consequence, this would allow storing of incorrect data which could disrupt the protocol's liveliness.

Recommendation

We recommend implementing validation for the provided `NonNativeRewardsItem` entries before storing them.

Status: Partially Resolved

53. Local height tracking could lead to incorrect ICQ results

Severity: Minor

Tracking local heights for interchain queries (ICQ) can potentially lead to incorrect results when ICQ data is queried before the execution of an interchain account action (ICA) on the remote chain, but the data is submitted after receiving the ICA acknowledgment (ACK).

This mismatch can cause inconsistencies in the data being returned and processed due to missing synchronization.

This issue has been independently reported by a third party.

Status: Externally Reported

54. Unhandled overflow

Severity: Minor

In `packages/puppeteer-base/src/state.rs:251-271`, the `reconstruct` function performs a conversion from `Decimal` to `u128`.

This conversion has the potential to overflow, which could result in a panic due to the `overflow-checks` flag being set.

This issue has been independently reported by a third party.

Status: Externally Reported

55. `LAST_PUPPETEER_RESPONSE` is not being reset on `Undelegate` error

Severity: Informational

In `contracts/core/src/contract.rs:712-750`, the `execute_tick_unbonding` function checks the response from the puppeteer hook, performing the necessary bookkeeping on undelegation success and error.

However, if the undelegation failed, the `LAST_PUPPETEER_RESPONSE` is not being reset, allowing the finite state machine (FSM) to execute a new tick, and act on the outdated puppeteer response.

Recommendation

We recommend removing `LAST_PUPPETEER_RESPONSE` upon an `Undelegate` error, even though this cannot cause any inconsistent state.

Status: Resolved

56. The `IBCTransfer` callback handler is not being used

Severity: Informational

In `contracts/puppeteer/src/contract.rs:339-350`, the function `execute_ibc_transfer` constructs an `IBCTransfer` message with a callback to `SudoPayload`, instead of `IBCTransfer` which is never used by protocol's contracts.

Recommendation

We recommend constructing the message to callback to the `IBCTransfer` handler.

Status: Resolved

57. Callback execution without access control

Severity: Informational

In `contracts/factory/src/contract.rs:115`, the `Callback` message is executed without enforcing any access control, allowing anyone to run it.

However, due to its callback nature, it should be executed solely by the contract itself.

Recommendation

We recommend implementing access controls to ensure only the contract itself can execute this function.

Status: Resolved

58. Missing use of checked math functions to prevent overflow

Severity: Informational

Math operations performed within the contracts in scope do not leverage checked math functions to guard against overflow.

Since `overflow-checks` are enabled in the main `Cargo.toml` file, this does not pose an immediate security concern since overflows would result in panics.

However, adopting `checked` functions would allow better error handling.

Recommendation

We recommend using checked math functions to enhance error handling of overflows.

Status: Acknowledged

59. "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: Resolved

60. Incorrect event emissions

Severity: Informational

In `contracts/factory/src/contract.rs:237`, the event emission in the response of the `execute_update_config` function is inaccurate. It emits an event with the text `execute-proxy-call`, which is incorrect as the function does not involve a proxy call.

Also, in `contracts/core/src/contract.rs:808`, in `execute_update_config`, due to a key collision in line `825`, the `puppeteer_timeout` will incorrectly update the `puppeteer_contract` event key.

Recommendation

We recommend updating the event emission text in the `execute_update_config` function from `execute-proxy-call` to a term like `execute-update-config`.

We also recommend updating `puppeteer_contract` to `puppeteer_timeout` in `contracts/core/src/contract.rs:825`.

Status: Resolved

61. Resolve TODOs

Severity: Informational

There are multiple TODOs across the codebase that indicate that development may not be finished and the codebase might not be ready for production deployment:

- In `packages/puppeteer-base/src/execute.rs:146`.
- In `packages/base/src/msg/puppeteer.rs:277`.
- In `packages/base/src/state/core.rs:64-66`.
- In `packages/puppeteer-base/src/execute.rs:146`.
- In `contracts/core/src/contract.rs:1444`.
- In `contracts/puppeteer/src/contract.rs:460-462`.
- In `contracts/validators-set/src/contract.rs:152, 193, 241`.

Recommendation

We recommend resolving or removing all TODOs before deployment.

Status: Acknowledged

62. Misleading function name and documentation

Severity: Informational

In `packages/helpers/src/pause.rs:29-36`, in contrast to the function name, `assert_paused` is used to ensure that the contract state is not paused.

Also, the function documentation mentions that the function is used to ensure that the caller account is the contract owner, which is misleading.

Recommendation

We recommend renaming the function to `assert_not_paused` and updating the function documentation accordingly.

Status: Resolved

63. Miscellaneous Comments

Severity: Informational

- The `unbond_batch_switch_time` is updated twice in `contracts/core/src/contract.rs:873-886`.
- Hardcoded mock address in `contracts/factory/src/contract.rs:469`.
- Validate the addresses in `contracts/puppeteer/src/contract.rs:557` and `558`.

Recommendation

We recommend addressing the aforementioned miscellaneous issues.

Status: Partially Resolved

Appendix

1. Chain data for [“Missing funds in the token contract required to pay the DenomCreationFee prevents the protocol from being deployed”]

```
./neutrond-linux-amd64 q tokenfactory params --chain-id "neutron-1"
params:
  denom_creation_fee: []
  denom_creation_gas_consume: "0"
  fee_collector_address: ""
```