Security Audit Report

Structured Private Deposit Minter and Oracle Contracts

v1.0

September 19, 2025

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 GmbH has been engaged by Droplet Labs Ltd to perform a security audit of Structured Private Deposit Minter and Oracle 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/structured-org/maxbtc-neutron

Commit	`Core: f11b7dfd047626a935030416a2c86f5e0bfaebdd`
	`Oracle: 6486d1d893f00311db6e9830acd7d5cd31129ed7`

Scope	All contracts and packages were in scope except for the mock exchange rate provider.
Fixes verified at commit	`f4e3f62b5e97fc2e35d08f105f067942d34e8198` 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

The core repository implements minting of maxBTC tokens backed by BTC deposits, featuring an automated fee collection mechanism that extracts performance fees based on exchange rate gains. The system includes allowlist controls for deposits, exchange rate oracle integration, and automated deposit forwarding to Ethereum via IBC, with the core functionality distributed across multiple smart contracts, including a core minting contract, fee collector, exchange rate provider, and allowlist manager.

The aum oracle repository implements a cross-chain Assets Under Management messaging system that aggregates financial data from multiple sources, including Binance and Jupiter, and then transmits this aggregated information to Neutron blockchain smart contracts for on-chain AUM calculations and reporting.

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**, **Partially Resolved**, 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 codebase has moderate to high complexity with a sophisticated cross-chain architecture involving and implementing custom consensus
Code readability and clarity	Medium-High	The code demonstrates good readability with clear naming conventions, well-structured interfaces, and good code commenting
Level of documentation	High	The provided notion documents thoroughly covered the codebase and provided all the necessary details to understand it.
Test coverage	Medium	Neutron-Core: 34.66% Oracle: 83.82% Aum Messenger: 7.6%

Summary of Findings

No	Description	Severity	Status
1	Race condition in Binance messenger data filtering causes potential crashes and data corruption	Critical	Resolved
2	`TOTAL_DEPOSITED` tracks minted amount instead of actual deposits, allowing deposit cap bypass	Critical	Resolved
3	Hardcoded test balance makes forwarder non-functional	Critical	Acknowledged
4	Silent failures lead to indefinite fund accumulation in forwarder	Major	Acknowledged
5	Outdated configuration used to compute the next round details	Major	Resolved
6	Previous owner retains privileges over the fee collector contract	Major	Acknowledged
7	Incorrect handling of `collection_period_seconds`	Major	Resolved
8	Poor secret management practices in `aum_messenger` deployment	Major	Acknowledged
9	Economic consensus manipulation	Major	Acknowledged
10	Potential division by zero in AUM calculation	Minor	Resolved
11	Hardcoded decimal divisor with TODO	Minor	Resolved
12	Gas DoS through unbounded oracle list	Minor	Acknowledged
13	Lack of immediate response mechanisms due to pending configuration updates	Minor	Acknowledged
14	Unvalidated consensus and price data valid periods	Minor	Resolved
15	Messenger list not validated for duplicates or empty lists	Minor	Resolved
16	Threshold not validated against the messenger set and zero values	Minor	Resolved
17	Sensitive information logged in the application config	Minor	Resolved

18	Updating the core contract address does not reset the state	Minor	Resolved
19	Unsafe update of `maxbtc_decimals` causes inconsistent configuration	Minor	Resolved
20	Users may receive significantly less maxBTC than expected due to the lack of slippage protection	Minor	Resolved
21	Centralization risks	Minor	Acknowledged
22	Single oracle failure causes complete system failure	Minor	Acknowledged
23	Division by zero vulnerability in exchange rate calculation due to zero `MOCKED_MAXBTC_SUPPLY`	Minor	Resolved
24	Inefficient `ALLOW_LIST` implementation may cause a denial of service	Minor	Acknowledged
25	Missing configuration validation	Informational	Acknowledged
26	Missing action attribute in publish response	Informational	Acknowledged
27	Timer is not stopped when the context is canceled	Informational	Acknowledged
28	Response code check occurs after unmarshalling	Informational	Acknowledged
29	Misleading event emission during ownership update	Informational	Resolved
30	Contracts should implement a two-step ownership transfer	Informational	Resolved
31	Indefinite shutdown wait could cause hanging process	Informational	Acknowledged
32	Contracts should emit detailed attributes	Informational	Acknowledged
33	Misleading success response for premature flush attempts	Informational	Acknowledged
34	`execute_update_config` should validate that paused state actually changes	Informational	Acknowledged
35	deposit_flush_period should have a minimum value to prevent spam	Informational	Acknowledged
36	flush and collection parameters should have a minimum value to prevent spam	Informational	Acknowledged

37	Incomplete implementation with test placeholders	Informational	Acknowledged

Detailed Findings

1. Race condition in Binance messenger data filtering causes potential crashes and data corruption

Severity: Critical

In the `fetchBinanceData` function in `aum_messenger/messenger/binance/types.go:255-260`, the code modifies the `umPositions` slice while iterating over it. This creates a race condition where removing elements during iteration causes index shifting, potentially leading to skipped elements, index out of bounds errors, and application crashes. This could result in incomplete AUM data being submitted to contracts, leading to incorrect calculations and potential financial losses.

Recommendation

We recommend refactoring the filtering logic to create a new slice instead of modifying the existing one during iteration.

Status: Resolved

2. `TOTAL_DEPOSITED` tracks minted amount instead of actual deposits, allowing deposit cap bypass

Severity: Critical

In the `execute_deposit` function in `contracts/maxbtc-neutron-core/src/contract.rs:251-253`. The `TOTAL_DEPOSITED` state variable tracks the total amount of `maxBTC` tokens minted rather than the actual deposit assets received. This creates a discrepancy between the intended deposit cap and the actual amount of underlying assets that can be deposited, allowing users to exceed the intended deposit cap by significant amounts depending on the fee structure.

Recommendation

We recommend updating the `TOTAL_DEPOSITED` tracking to use the actual deposit amount instead of the minted amount to ensure the deposit cap is properly enforced on the underlying asset value.

Status: Resolved

3. Hardcoded test balance makes forwarder non-functional

Severity: Critical

In `forwarder_executor/src/main.rs:52`, function `run_cycle` contains a hardcoded balance value that overrides the actual queried balance, making the forwarder service completely non-functional.

Impact:

- Funds accumulate in contract without being forwarded
- If actual balance < 100000: transactions fail
- If actual balance > 100000: excess funds stuck
- Complete failure of cross-chain forwarding

Recommendation

We recommend removing `line 52` in `forwarder_executor/src/main.rs` entirely.

Status: Acknowledged

4. Silent failures lead to indefinite fund accumulation in forwarder

Severity: Major

In `forwarder_executor/src/main.rs:27-29`, the forwarder service silently continues after failures without any error recovery mechanism. When `run_cycle` fails, it only logs the error and continues to the next cycle, causing funds to accumulate indefinitely in the contract with no operator alerts.

Impact:

- Failed forwarding attempts are silently ignored
- Funds accumulate in the contract indefinitely
- No alerts to operators about failures
- Manual intervention required to recover stuck funds

Recommendation

We recommend implementing retry logic with exponential backoff for transient errors, alerting after consecutive failures, and persisting failed transaction details for manual recovery when automated retries exceed limits.

Status: Acknowledged

5. Outdated configuration used to compute the next round details

Severity: Major

In contracts/binance-aum-receiver/src/contract.rs:100, the execute_publish_data function uses consensus_config.round_length to calculate the next_round variable, which configuration is retrieved from CONSENSUS_STATE.config in line 74.

The issue is that CONSENSUS_STATE.publish_data may have updated CONSENSUS_STATE.config from the CONSENSUS_STATE.pending_config state (see packages/consensus/src/consensus.rs:224-228), causing the previously loaded consensus_config to be outdated.

Consequently, the next_round and next_round_timestamp events emitted will be incorrect. This affects the aum_messenger, which relies on these values for round scheduling in aum_messenger/client/neutron/client.go:176-183.

This issue also affects contracts/jupiter-aum-receiver/src/contract.rs:166.

Recommendation

We recommend retrieving CONSENSUS_STATE.config after the publish_data call to ensure the latest configuration values are used when computing next_round.

Status: Resolved

6. Previous owner retains privileges over the fee collector contract

Severity: Major

In contracts/maxbtc-neutron-core/src/contract.rs:68-72, when instantiating the fee collector contract via WasmMsg::Instantiate2, the contract migration admin and the FeeCollectorInstantiateMsg.owner are set to the maxbtc-neutron-core contract's owner. This is problematic because the maxbtc-neutron-core contract's owner can be updated via execute_update_config, but the previous owner still retains the following privileges:

- Contract migration permissions for the fee collector contract. This allows the previous owner to update the code ID for the fee collector contract, such as introducing a backdoor to withdraw funds.
- FeeCollectorInstantiateMsg.owner role, allowing them to call privileged functions such as execute_claim and execute_update_config in the fee collector contract to withdraw funds or issue configuration changes.

Consequently, the previous owner still retains control over the fee collector, even after ownership is changed in the core contract.

Recommendation

We recommend setting the fee collector contract’s admin and owner to the maxbtc-neutron-core contract address. Additionally, consider introducing a privileged entry point that allows the current core contract owner to migrate or update the fee collector contract’s configuration via the maxbtc-neutron-core contract.

Status: Acknowledged

7. Incorrect handling of collection_period_seconds

Severity: Major

In contracts/maxbtc-neutron-fee-collector/src/contract.rs:44, the collection_period_seconds field is set directly from msg.collection_period_seconds without converting hours into seconds. This is incorrect because, according to the documentation in contracts/maxbtc-neutron-core/src/msg.rs:117, the input value represents the duration in hours (*The duration in hours for each fee collection period*).

Consequently, the collection period will be configured 3600 times shorter than intended. For example, if collection_period_seconds is set to 1 (which expects one hour), the contract will incorrectly interpret it as 1 second.

Recommendation

We recommend updating the instantiation logic to convert the msg.collection_period_seconds field from hours into seconds, similar to contracts/maxbtc-neutron-fee-collector/src/contract.rs:225.

Status: Resolved

8. Poor secret management practices in aum_messenger deployment

Severity: Major

In the aum_messenger service configuration and deployment setup, sensitive credentials including Binance API keys, API secrets, and Cosmos wallet mnemonic phrases are stored in plain text YAML configuration files that are directly mounted into Docker containers. The docker-compose.yml shows the config file is mounted as a volume, and the config.yaml.default template contains fields for binance_api_key, binance_api_secret, and clients.neutron.mnemonic. This approach exposes sensitive credentials at the filesystem level and provides no encryption, secret rotation, or access controls, making the system vulnerable to credential theft if the container or host filesystem is compromised.

Recommendation

We recommend implementing proper secret management practices including using environment variables, Docker secrets, or external secret management services to securely handle sensitive credentials instead of storing them in plain text configuration files.

Status: Acknowledged

9. Economic consensus manipulation

Severity: Major

In `packages/consensus/src/consensus.rs:395-432`, specifically the `consensus_on_items` function and the subset selection logic in lines `411-431`, an attacker controlling the threshold number of messengers can manipulate consensus prices through coordinated submissions.

The vulnerability lies in how the function finds the largest valid subset within delta tolerance in lines `413-426`. By having messengers submit values at the edge of the delta tolerance (e.g., if delta is 5%, submit values 4.9% apart), the attacker can gradually shift the consensus price over multiple rounds while all submissions appear legitimate.

For example, with a 5% delta and 10 rounds, prices could be shifted by up to 50% cumulatively, potentially affecting millions in minting ratios.

Recommendation

We recommend implementing stake-based security where messengers must lock collateral that can be slashed for provable manipulation, adding economic disincentives.

Status: Acknowledged

10. Potential division by zero in AUM calculation

Severity: Minor

In `contracts/twaer/src/contract.rs:289-292`, the calculation `Decimal::from_ratio(aum, maxbtc_supply)` will panic if `maxbtc_supply` is zero, causing the contract to fail.

Recommendation

We recommend adding a check to ensure `maxbtc_supply` is non-zero before performing the division.

Status: Resolved

11. Hardcoded decimal divisor with TODO

Severity: Minor

In `aum_messenger/messenger/jupiter/types.go:77`, the AUM calculation uses a hardcoded divisor of `1000000` with a TODO comment indicating uncertainty. An incorrect divisor would cause massive calculation errors.

Recommendation

We recommend verifying the correct decimal places for each token and implementing dynamic decimal handling based on token configuration.

Status: Resolved

12. Gas DoS through unbounded oracle list

Severity: Minor

In `contracts/twaer/src/contract.rs`, the TWAER contract allows unlimited oracle additions through the `update_config` function in lines `63-99` and queries all oracles in `get_aum` function in lines `340-354`, creating a gas exhaustion vector. An attacker can create a big number of oracle contracts, and if the admin is compromised, each query could exceed the block limit. As a result, TWAER is permanently DoS'd, and cannot calculate exchange rates

Recommendation

We recommend implementing comprehensive limits on the number of oracles.

Status: Acknowledged

13. Lack of immediate response mechanisms due to pending configuration updates

Severity: Minor

In `contracts/binance-aum-receiver/src/contract.rs:132-150`, when updates are made to `CONSENSUS_STATE.config`, they are first stored in `pending_config` and only applied in the next consensus round (see packages/consensus/src/consensus.rs:140). This is problematic because there is a delay before critical changes take effect.

For example, if a messenger in consensus_config.messengers is compromised, it cannot be removed immediately. The compromised messenger would still be able to manipulate AUM calculations for the duration of the current round.

Recommendation

We recommend introducing an emergency mechanism to bypass the delayed pending_config process for critical updates, such as immediately removing compromised messengers.

Status: Acknowledged

14. Unvalidated consensus and price data valid periods

Severity: Minor

In contracts/binance-aum-receiver/src/state.rs:175-180, when updating consensus_data_valid_period and price_data_valid_period, the update_config function does not validate that these values are greater than zero. In contrast, the Jupiter contract enforces this validation by invoking Config::validate (see packages/jupiter-aum-common/src/types.rs:27-38).

Consequently, zero values for consensus_data_valid_period and price_data_valid_period could be stored in the Binance AUM Receiver contract.

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

Recommendation

We recommend enforcing the same validation in the Binance contract.

Status: Resolved

15. Messenger list not validated for duplicates or empty lists

Severity: Minor

In a few instances of the codebase, when updating messengers, the logic does not check for duplicate entries or ensure that the list is non-empty.

This occurs in the following instances:

- contracts/binance-aum-receiver/src/contract.rs:23-27
- contracts/binance-aum-receiver/src/contract.rs:131-138

- `contracts/jupiter-aum-receiver/src/contract.rs:46-50`
- `contracts/jupiter-aum-receiver/src/contract.rs:110-116`

Consequently, if duplicates are present, the same messenger could be counted multiple times during consensus formation. On the other hand, an empty list would effectively disable messenger validation, breaking consensus requirements.

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

Recommendation

We recommend enforcing deduplication of messenger addresses and validating that the list is not empty in the instances mentioned above.

Status: Resolved

16. Threshold not validated against the messenger set and zero values

Severity: Minor

In a few instances of the codebase, when updating the threshold, the logic does not check whether the threshold is non-zero and does not exceed the number of configured messengers.

This occurs in the following instances:

- `contracts/binance-aum-receiver/src/contract.rs:30`
- `contracts/binance-aum-receiver/src/contract.rs:139-141`
- `contracts/jupiter-aum-receiver/src/contract.rs:51`
- `contracts/jupiter-aum-receiver/src/contract.rs:117-119`

Consequently, an invalid threshold could be stored in the consensus configuration, potentially making consensus formation impossible. On the other hand, a zero-value threshold would cause incorrect consensus formation.

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

Recommendation

We recommend validating that the threshold is greater than zero and less than or equal to the number of messengers in the instances mentioned above.

Status: Resolved

17. Sensitive information logged in the application config

Severity: Minor

In aum_messenger/main.go:46, the application logs the entire configuration object. This is problematic because it includes sensitive fields such as BinanceApiKey, BinanceApiSecret, and Clients.Neutron.Mnemonic (see aum_messenger/conf.go:34-37 and aum_messenger/utils/cosmos_client.go:50).

Consequently, these secrets are exposed in logs, potentially leading to unauthorized access to Binance and Neutron services.

Recommendation

We recommend redacting sensitive fields when logging configuration.

Status: Resolved

18. Updating the core contract address does not reset the state

Severity: Minor

In contracts/maxbtc-neutron-fee-collector/src/contract.rs:210, when the core contract address is updated in execute_update_config, the State that records the last_collection_timestamp and last_exchange_rate is not updated. This results in the execute_collect_fee logic to potentially use stale values that originated from the previous core contract.

Consequently, the execute_collect_fee function will miscalculate the fees after updating the core contract address.

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

Recommendation

We recommend performing one of the following recommendations:

- If updating the core contract is a required feature, consider updating the State upon updating the core contract by querying the latest exchange rate and setting last_collection_timestamp to the current block time, similar to contracts/maxbtc-neutron-fee-collector/src/contract.rs:51-59.
- If updating the core contract is not a needed feature, consider removing the ability to update the core contract from execute_update_config.

Status: Resolved

19. Unsafe update of `maxbtc_decimals` causes inconsistent configuration

Severity: Minor

In `contracts/maxbtc-neutron-fee-collector/src/contract.rs:229`, the `execute_update_config` function allows the contract owner to update the `maxbtc_decimals` value. This is problematic because the core collector contract does not permit this field to be modified, which is intended to prevent calculation errors.

Consequently, allowing updates for `maxbtc_decimals` could desynchronize configuration values between the fee collector and the core collector, leading to incorrect fee minting calculations.

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

Recommendation

We recommend disallowing the contract owner from updating `maxbtc_decimals` in `execute_update_config`.

Status: Resolved

20. Users may receive significantly less maxBTC than expected due to the lack of slippage protection

Severity: Minor

In `contracts/maxbtc-neutron-core/src/contract.rs:214-262`, `ExecuteMsg::Deposit` function has no slippage protection. While the transaction itself is atomic, users cannot specify a minimum amount of maxBTC they are willing to accept. The exchange rate can change between when a user decides to deposit (after checking rates) and when their transaction is executed on-chain.

For instance, user queries rate (1 BTC = 10 maxBTC), submits transaction, but oracle updates before execution, resulting in only 5 maxBTC received.

Recommendation

We recommend adding `min_amount_out` parameter to the deposit function and revert if `minted_amount < min_amount_out`.

Status: Resolved

21. Centralization risks

Severity: Minor

The protocol has excessive centralization with single owner addresses controlling critical functions. While these require privileged access and are not exploitable by external attackers, they represent significant trust assumptions.

Core contract owner:

- Can redirect all protocol fees
- Can pause/unpause at will
- Can remove deposit caps
- Can change core parameters without time delays

Fee collector contract owner:

- Can set fee percentage up to 99% without bounds
- No time delays or multi-sig requirements

Forwarder service operator:

- Single service instance with no redundancy
- Service downtime delays fund forwarding (though anyone can call FlushDeposits)
- Holds private keys for gas payments
- No configuration validation - invalid configs can cause service failures

Recommendation

We recommend implementing multi-sig or DAO governance for all owner functions, bounds checking for all configurable values, time delays for critical parameter changes, and progressive decentralization roadmap.

Status: Acknowledged

22. Single oracle failure causes complete system failure

Severity: Minor

In `contracts/twaer/src/contract.rs:344-349`, a single oracle failure causes complete TWAER failure, creating a critical single point of failure that could make the entire oracle system unavailable even if other oracles are functioning correctly.

Recommendation

We recommend implementing fallback mechanisms using last known values or quorum-based calculations to maintain system availability during partial oracle failures.

Status: Acknowledged

23. Division by zero vulnerability in exchange rate calculation due to zero `MOCKED_MAXBTC_SUPPLY`

Severity: Minor

In the `twaer` contract, the `MOCKED_MAXBTC_SUPPLY` constant is not validated to ensure it's non-zero, which can cause a panic during exchange rate calculations in the `from_ratio` function. While this case is unlikely, it should still be validated to be greater than zero.

Recommendation

We recommend adding a validation to ensure `MOCKED_MAXBTC_SUPPLY` is always greater than zero during contract instantiation

Status: Resolved

24. Inefficient `ALLOW_LIST` implementation may cause a denial of service

Severity: Minor

In `contracts/maxbtc-neutron-allow-list/src/state.rs:4`, the `ALLOW_LIST` state is stored as an `Item<Vec<Addr>>`. This is problematic because any updates or queries to this state require loading and iterating over the entire vector, which may cause an out-of-gas error.

For example, in `ExecuteMsg::UpdateAllowList`, the entry point validates and saves the entire list in a single transaction. If the list is extensive (e.g., many KYC-approved addresses), the transaction may fail due to gas limits, causing a denial of service.

This also affects the `QueryMsg::IsAddressAllowed` and `QueryMsg::AllowList` queries, as all the addresses in the `ALLOW_LIST` are iterated, resulting in `O(n)` complexity and potential out-of-gas errors as the list grows.

Consequently, large allow lists may become unmanageable, preventing successful updates and queries. This reduces scalability and may prevent new users from being added to the allow list once it grows beyond gas constraints.

Additionally, this design poses maintainability concerns as updating the allowlist requires a complete overwrite of the existing allow list even to add one address. If the allowlist is very large, requiring all the addresses to be provided in a single vector in a single transaction could exceed transaction size limits, could result in errors or mistakes, and in general is much more difficult to manage effectively.

Recommendation

We recommend performing the following recommendations:

- Refactor the `ALLOW_LIST` to use [`Map<Addr, bool>`](). This allows `QueryMsg::IsAddressAllowed` to be optimized with `O(1)` lookups using `ALLOW_LIST.has(storage, addr)`, while `ExecuteMsg::UpdateAllowList` can be processed within several transactions.
- Implement pagination mechanisms in the `QueryMsg::AllowList` to allow entries to be fetched in batches.

Status: Acknowledged

25. Missing configuration validation

Severity: Informational

In `aum_messenger/conf.go`, configuration values are loaded without any validation of required fields, value ranges, or format correctness. This could lead to runtime failures or unexpected behavior.

Recommendation

We recommend implementing comprehensive validation for all configuration fields including URL formats, positive integer values for timeouts, and required field presence checks.

Status: Acknowledged

26. Missing action attribute in publish response

Severity: Informational

In `contracts/binance-aum-receiver/src/contract.rs:83`, the `execute_publish_data` function creates a new `Response` after calling `CONSENSUS_STATE.publish_data`, but it does not include an `add_attribute("action", "publish_consensus")`. In contrast, `contracts/jupiter-aum-receiver/src/contract.rs:150` sets this attribute for better event tracking.

Recommendation

We recommend adding `res = res.add_attribute("action", "publish_consensus")` to align with the behavior in the `jupiter-aum-receiver` contract.

Status: Acknowledged

27. Timer is not stopped when the context is canceled

Severity: Informational

In `aum_messenger/messenger/messenger.go:77-97`, the `RunMessenger` function uses `time.NewTimer(timeTillNextRound).C` directly inside the `select` statement. If `ctx.Done()` fires before the timer, the timer will continue running in the background until expiry.

This wastes memory as the timer remains in the runtime's timer heap and may increase garbage collector (GC) pressure.

Recommendation

We recommend creating the timer explicitly and stopping it if `ctx.Done()` is reached first:

```go
timer := time.NewTimer(timeTillNextRound)
defer timer.Stop()

select {
case <-timer.C:
    // ...
case <-ctx.Done():
    if !timer.Stop() {
        <-timer.C
    }
    return
}
```

Status: Acknowledged

28. Response code check occurs after unmarshalling

Severity: Informational

In `aum_messenger/utils/cosmos_client.go:293-307`, the `calculateGas` function unmarshals `res.Response.Value` into `simRes` before checking whether `res.Response.Code` indicates a failure.

This ordering means an invalid or failed response may still be unmarshalled unnecessarily, which does not align logically since simRes depends on a valid res.

Recommendation

We recommend moving the if res.Response.Code != 0 check before the unmarshalling step.

Status: Acknowledged

29. Misleading event emission during ownership update

Severity: Informational

In contracts/maxbtc-neutron-allow-list/src/contract.rs:46-51, when handling ExecuteMsg::UpdateOwnership, the contract emits an event attribute ("new_owner", info.sender). This is misleading because not all ownership update actions result in a new owner. Specifically, when RenounceOwnership is executed, the ownership is removed entirely, yet the emitted event incorrectly logs the sender as the "new_owner".

Recommendation

We recommend emitting only the action (e.g., "renounce_ownership", "transfer_ownership").

Status: Resolved

30. Contracts should implement a two-step ownership transfer

Severity: Informational

The contracts within the scope of this audit allow the current owner to execute a one-step ownership transfer. While this is common practice, it presents a risk for the ownership of the contract to become lost if the owner transfers ownership to an incorrect address.

A two-step ownership transfer will allow the current owner to propose a new owner, and then the account that is proposed as the new owner may call a function that will allow them to claim ownership and actually execute the config update.

The following instances do not implement a two-step ownership transfer:

- contracts/maxbtc-neutron-core/src/contract.rs:177
- contracts/binance-aum-receiver/src/state.rs:173
- contracts/jupiter-aum-receiver/src/contract.rs:89-91
- contracts/twaer/src/contract.rs:82-85

Recommendation

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

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

Status: Resolved

31. Indefinite shutdown wait could cause hanging process

Severity: Informational

In the `main` function in `aum_messenger/main.go:207`, the `wg.Wait()` call waits indefinitely for all messenger goroutines to finish. If any messenger gets stuck or hangs during shutdown, the process will never terminate gracefully.

Recommendation

We recommend implementing a timeout mechanism for graceful shutdown using `context.WithTimeout`.

Status: Acknowledged

32. Contracts should emit detailed attributes

Severity: Informational

In all three of the oracle contracts - `binance-aum-receiver`, `jupiter-aum-receiver`, and `twaer` the instantiation functions lack detailed event attributes that would provide important information for indexing, monitoring, and debugging purposes.

Recommendation

We recommend adding detailed event attributes to all three contract instantiation functions.

Status: Acknowledged

33. Misleading success response for premature flush attempts

Severity: Informational

In the `execute_flush_deposits` function in `contracts/maxbtc-neutron-core/src/contract.rs:281-287` when called before the flush period has elapsed, the function returns a successful response instead of an error. This impacts user experience and provides misleading feedback.

Recommendation

We recommend returning an error if the flush attempt fails.

Status: Acknowledged

34. `execute_update_config` should validate that paused state actually changes

Severity: Informational

In the `execute_update_config` function in `contracts/maxbtc-neutron-core/src/contract.rs:172-175`. When updating the paused state, the function should validate that the new value is different from the current value to prevent unnecessary state updates and provide clearer feedback to users.

Recommendation

We recommend adding a check to ensure the paused state actually changes before updating it, returning an error if the new value is identical to the current value.

Status: Acknowledged

35. `deposit_flush_period` should have a minimum value to prevent spam

Severity: Informational

In the `execute_flush_deposits` function in `contracts/maxbtc-neutron-core/src/contract.rs:279-287`. The `deposit_flush_period` configuration lacks a minimum value constraint, which could allow it to be set to 0 or very low values. This would enable the flush endpoint to be spammed, potentially sending many small amounts to the deposit forwarder contract.

Recommendation

We recommend enforcing a minimum value constraint for `deposit_flush_period` during configuration updates to prevent spam and to ensure reasonable flush intervals.

Status: Acknowledged

36. Flush and collection parameters should have a minimum value to prevent spam

Severity: Informational

In the execute_flush_deposits function in contracts/maxbtc-neutron-core/src/contract.rs:279-287. The deposit_flush_period configuration lacks a minimum value constraint, which could allow it to be set to 0 or very low values. This would enable the flush endpoint to be spammed, potentially sending many small amounts to the deposit forwarder contract.

Additionally, the collection_period_seconds in contracts/maxbtc-neutron-fee-collector/src/contract.rs:224 should have minimum value enforced.

Recommendation

We recommend enforcing a minimum value constraints for deposit_flush_period and collection_period_seconds during configuration updates to prevent spam and to ensure reasonable flush intervals.

Status: Acknowledged

37. Incomplete implementation with test placeholders

Severity: Informational

In aum_messenger/client/solana/client.go:24-29, the codebase contains TODO comments and test implementations that could lead to incorrect oracle data being used in production. The global mutable variable lacks synchronization and functions return hardcoded test values instead of actual blockchain queries, potentially causing financial miscalculations.

Recommendation

We recommend completing all TODO implementations and removing test code before deployment.

Status: Acknowledged