Security Audit Report

Structured CosmWasm and Solana Updates

v1.0

December 24, 2025

Table of Contents

License

Disclaimer

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 to perform a security audit of Structured CosmWasm and Solana Updates.

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-solana`
Label	`maxbtc-solana`
Commit	`e10d6b4d3a187f0e75919f6c7589ee05a5bc131f`
Scope	The scope is restricted to the following programs: • `programs/jupiter-helper` • `programs/wormhole-helper` • `programs/waitosaur`

Fixes verified at commit	5515de99f2dc0576d6fbe715194223b26bc1241d Note that only fixes to the issues described in this report have been reviewed at this commit. Any further changes such as additional features have not been reviewed.

Repository	https://github.com/structured-org/aum-oracle
Label	aum-oracle
Commit	2ed17e99debca94b712758484c81f46efd3f4e9a
Scope	The scope is restricted to the following contracts: ● contracts/waitasaurus
Fixes verified at commit	cd4e2f36bddaf3dd8e8f04af8144415d9e44178f Note that only fixes to the issues described in this report have been reviewed at this commit. Any further changes such as additional features have not been reviewed.

Repository	https://github.com/structured-org/maxbtc-neutron
Label	maxbtc-neutron
Commit	9721fc1557cb6d56aab2b201d16e1c9f71f27829
Scope	The scope is restricted to the following contracts: ● contracts/maxbtc-neutron-allow-list ● contracts/maxbtc-neutron-core ● contracts/maxbtc-neutron-factory ● contracts/maxbtc-neutron-fee-collector ● contracts/maxbtc-neutron-token ● contracts/maxbtc-neutron-waitosaur-holder ● contracts/maxbtc-neutron-withdrawal-manager ● packages
Fixes verified at commit	0c44e06c3a18a9f47956392d75d68daf7b1f3cc1

	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 maxBTC protocol coordinates a cross-chain yield strategy and tokenization system spanning Neutron, Solana, Binance, and Ethereum, where Neutron mints and burns the maxBTC token while the strategy legs on Binance and Solana handle asset deployment, hedging, and rebalancing.

Its architecture relies on oracles and specialized "waitosaur" contracts to ensure AUM correctness by explicitly marking periods when assets are in transit and therefore non-observable: holder waitosaurs receive and forward bridged assets on Neutron or Solana, while observer waitosaurs monitor expected balances on Binance.

A core finite-state machine governs deposit and withdrawal flows through an operator-triggered tick function, orchestrating actions such as initiating bridging, delegating funds to Ceffu, managing Jupiter Perpetuals entry and exit via helper programs, and synchronizing backend operations with on-chain state transitions.

Because assets move across multiple opaque domains and cooldown periods (e.g., Ceffu withdrawals), the system enforces staged processes where each leg only proceeds once the corresponding waitosaur is unlocked, ensuring consistency between strategy execution and token supply accounting despite partial off-chain components and multisig-managed controls.

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 protocol spans multiple chains, contract frameworks, and programming languages, and requires tight coordination with off-chain components and closed-source third-party protocols.
Code readability and clarity	**Medium-High**	-
Level of documentation	**Medium**	The protocol is generally well documented, though its integrations with third-party protocols are not thoroughly described.
Test coverage	**Medium**	The test coverage computed with `cargo tarpaulin` for `maxbtc-solana` is `0%`, though integration tests are present. The test coverage computed with `cargo tarpaulin` for `aum-oracle` is `85.29%`. The test coverage computed with `cargo tarpaulin` for `maxbtc-neutron` is `40.34%`.

Summary of Findings

No	Description	Severity	Status
1	Missing oracle staleness validation before using price data in Solana programs	Major	Acknowledged
2	Lack of AUM Oracle state validation can lead to incorrect deposit flow tracking	Major	Resolved
3	Permissionless burn enables denom recreation failure and withdrawal DoS	Major	Resolved
4	Oversized unlocks allow overpayment and cross-batch imbalance in claim logic	Major	Acknowledged
5	Waitasaurus does not validate oracle data timestamp	Major	Resolved
6	Minted fee amount can be manipulated in both directions	Major	Acknowledged
7	Deposit flushing is vulnerable to Denial-of-Service attacks	Major	Acknowledged
8	Missing rent-exemption checks for SOL withdrawals	Major	Resolved
9	Sanctioned addresses can purchase and redeem MaxBTC	Major	Resolved
10	Lack of exchange rate liveliness checks allows over-minting of MaxBTC	Major	Resolved
11	Missing mint alignment check may cause a miscalculated AUM	Major	Resolved
12	Unrestricted access enables sandwich attack on the process handler	Major	Acknowledged
13	Existing MaxBTC issuer contract migration does not transfer current deposits	Major	Resolved
14	Using the vault account as the Wormhole bridge fee payer allows griefing by fee draining	Major	Resolved
15	Unchecked `unwrap` on optional cap field causes panic	Major	Resolved
16	Misleading use of `TOTAL_DEPOSITED` leads to	Major	Resolved

	protocol lock		
17	No validation for lock amount allows negative and zero values	Minor	Resolved
18	Fixed Wormhole Helper configuration cannot be updated to match Wormhole fee changes	Minor	Resolved
19	Configuration changes while locked can prevent unlocking	Minor	Resolved
20	Potential division by zero in liquidity calculations	Minor	Resolved
21	Liquidity operation fees can be set up to 100%	Minor	Acknowledged
22	Missing validation on collection period allows invalid or inconsistent configuration	Minor	Resolved
23	Unverified trust assumption in Jupiter Perpetuals integration	Minor	Acknowledged
24	Permissionless bridge function allows griefing attacks	Minor	Resolved
25	Owner can withdraw assets regardless of the lock state	Minor	Resolved
26	Owner can bypass lock state using `reset_locker`	Minor	Acknowledged
27	Core contract reassignment without namespacing risks withdrawal state corruption	Minor	Resolved
28	Protocol centralization due to off-chain operator dependency	Minor	Acknowledged
29	Fragile local calculation of output amount depends on external program logic	Minor	Resolved
30	Conflicting configuration flags allow ambiguous processor update	Minor	Resolved
31	Zero-value lock permitted due to missing aggregate amount validation	Minor	Resolved
32	Single-block instant withdrawal can stifle AUM growth	Minor	Acknowledged
33	Manual allow list growth can cause deposit Denial-of-Service	Minor	Resolved
34	Lack of validation for KYC service addresses allows for deposit Denial-of-Service	Minor	Resolved

35	Lack of deposit cost validation can lead to a subtraction overflow	Minor	Resolved
36	Lack of pagination for finalized batches query may lead to out-of-gas error	Minor	Resolved
37	Contracts allow for downgrading without changing the contract version	Minor	Acknowledged
38	Contracts instantiated by the maxbtc-neutron-factory contract cannot be upgraded	Minor	Acknowledged
39	Missing pubkey validation in Wormhole Helper initialization enables misconfiguration	Minor	Resolved
40	Missing pubkey validation in Waitosaur initialization enables misconfiguration	Minor	Resolved
41	Configuration updates allowed while the Waitosaur is locked may lead to an inconsistent state	Minor	Resolved
42	Unvalidated recipient address in claim execution may lead to misrouted funds	Minor	Resolved
43	Inaccessible `SetTokenMetadata` entrypoint leaves token without metadata	Minor	Acknowledged
44	Inefficient ZkMe error handling	Informational	Acknowledged
45	Configurable discriminator could lead to incorrect message types	Informational	Resolved
46	Broad version range in migration logic may lead to unintended upgrades	Informational	Acknowledged
47	Missing event emissions reduce observability	Informational	Resolved
48	Contracts should implement a two-step ownership transfer	Informational	Resolved
49	Incomplete TODO comments	Informational	Resolved
50	Unused error variants increase codebase complexity	Informational	Resolved
51	Incorrect event emission on token minting	Informational	Resolved
52	Missing associated token account mint constraint	Informational	Resolved
53	Use of magic numbers increases maintenance costs	Informational	Resolved

54	Misleading error	**Informational**	**Resolved**
55	Dead code and misleading comments increase maintenance costs	**Informational**	**Resolved**

Detailed Findings

1. Missing oracle staleness validation before using price data in Solana programs

Severity: Major

In maxbtc-solana:packages/base/src/formulas.rs:139-186, the custody_aum function uses oracle price data without validating its freshness.

The code directly accesses doves_oracle.price and doves_oracle.expo but does not check when this price was last updated. Stale oracle prices could lead to incorrect AUM calculations.

An attacker could exploit stale oracle prices during market volatility to execute trades at favorable prices when the real market has moved significantly, manipulate pool AUM calculations for profit, or cause incorrect minimum output calculations in liquidity operations.

The Doves oracle struct (AgPriceFeed) includes a timestamp field that could be used for staleness validation, but this is never checked in the codebase.

Recommendation

We recommend implementing oracle staleness checks before using price data. Add a timestamp validation to ensure oracle prices are recent (e.g., within 5 minutes) by checking the oracle's timestamp field against the current time.

Status: Acknowledged

The client acknowledges the issue, stating that Jupiter Perpetuals already performs this validation. In case of stale prices, it throws a StaleOraclePrice error. However, this is not independently verifiable.

2. Lack of AUM Oracle state validation can lead to incorrect deposit flow tracking

Severity: Major

The execute_lock and execute_unlock functions in aum-oracle:contracts/waitasaurus/src/contract.rs coordinate asset locking based on spot balance data from an external AUM oracle. However, both functions omit checks that compromise the correctness of deposit tracking.

In execute_lock, the contract transitions to the Locked state without verifying that the oracle-reported spot balance of the configured asset is zero. This oversight permits the lock process to begin while residual balances from previous deposits remain, allowing prior or unrelated inflows to be misattributed to the current lock.

In `execute_unlock`, the contract checks only whether the current spot balance meets the locked threshold. It does not ensure that the oracle data was published after the lock event, risking the use of outdated or unrelated data to authorize unlocking. As a result, operations like rebalancing hedge positions could interfere with the spot balance, causing incorrect asset accounting and potentially double-counting previous deposits.

Recommendation

We recommend implementing stricter temporal or transactional verification to correlate the locked amount with actual on-chain or off-chain events during the lock period. This could involve timestamp validation, tracking delta changes in asset balances, or introducing unique transfer identifiers to confirm that the locked value was indeed transferred and held exclusively as part of the lock action.

Status: Resolved

3. Permissionless burn enables denom recreation failure and withdrawal DoS

Severity: Major

In `maxbtc-neutron:contracts/maxbtc-neutron-core/src/contract.rs:569-580`, the `execute_withdraw` function treats a zero bank supply of a batch-specific redemption denom as an indication that the denom has not yet been created.

Based on this assumption, it attempts to create the denom via a `CreateRedemptionToken` message when `amount.is_zero` is `true`. However, this logic is flawed due to its reliance on total supply as a proxy for token existence.

The associated token contract's `execute_burn` method is permissionless and allows any user to burn tokenfactory-minted redemption tokens. This enables an attacker to exploit the timing between denom creation and withdrawal by executing a minimal withdrawal to mint a small amount of the new redemption token and immediately burning it, either in the same transaction as a second message or via a lightweight helper contract. This leaves the token's denom technically existing, but with a total supply of zero.

Since `execute_withdraw` does not check whether the denom already exists and only evaluates if the supply is zero, all subsequent withdrawals for that batch will incorrectly attempt to recreate the denom. This causes `MsgCreateDenom` to fail and reverts the entire withdrawal transaction, resulting in a denial-of-service for all future withdrawals for that batch. The attack is inexpensive, easily automatable, and can be reliably executed atomically.

Recommendation

We recommend changing the denom existence check from a supply-based condition to an explicit check for denom registration or creation status.

Status: Resolved

4. Oversized unlocks allow overpayment and cross-batch imbalance in claim logic

Severity: Major

In `maxbtc-neutron:contracts/maxbtc-neutron-withdrawal-manager/src/contract.rs:106-164`, the `execute_claim` function calculates a user's BTC payout for a finalized batch using a pro-rata share of `batch.collected_amount`, adjusted by the amount already paid out.

However, the `collected_amount` itself is subject to inflation due to upstream logic in `contracts/maxbtc-neutron-core/src/contract.rs`, where `execute_tick_withdraw_pending` adds the waitosaur holder's `Locked { amount }` to the batch without capping it to the actual remaining need (`btc_requested - collected_amount`).

This uncapped addition means that if the locker/backend submits a `Lock` with an excessive amount or sweeps surplus funds, the batch becomes overfunded. Since the `Unlock` call then transfers the full balance to the withdrawal manager, the inflated `collected_amount` is finalized and used by `execute_claim` to compute payouts. This results in overpayment to early claimants in the batch, misallocation of excess funds, under-collateralization of subsequent batches, and overall degradation of claim fairness.

The core contract currently trusts both the declared lock amount and the actual funds sent without reconciling them to batch needs.

Recommendation

We recommend modifying the `execute_tick_withdraw_pending` function in the core contract to cap the locked amount added to a batch by the remaining unmet withdrawal requirement. Additionally, the withdrawal manager should validate that claims are bounded by the burned maxBTC amount, preventing overpayment due to inflated `collected_amount`.

Status: Acknowledged

5. Waitasaurus does not validate oracle data timestamp

Severity: Major

In `aum-oracle:contracts/waitasaurus/src/contract.rs:119-124`, the contract queries oracle data but does not validate the freshness of the data using the timestamp field.

The `ConsensusOutcome` structure contains a `timestamp` field indicating when consensus was reached, but this is never checked against the current block time.

Although the oracle contract may have its own staleness checks, the waitosaur contract trusts the oracle data completely without verifying data age, which could lead to unlock decisions based on outdated information.

Recommendation

We recommend implementing a configurable maximum age for oracle data and validating that the `timestamp` field in the oracle response is within acceptable bounds before making unlock decisions.

Status: Resolved

6. Minted fee amount can be manipulated in both directions

Severity: Major

In `maxbtc-neutron:contracts/maxbtc-neutron-fee-collector/src/contract.rs:306-346`, the `calculate_fee_to_mint` function derives a target exchange rate from the delta between the current time-weighted average exchange rate (TWAER) and the last TWAER, with a fee rate applied. It then computes the mint amount as `total_supply * ((current_rate / target_rate) - 1)`.

In lines `96-158`, the `execute_collect_fee` handler calls `calculate_fee_to_mint` using the spot MaxBTC total supply, before minting the returned amount to the configured fee recipient in the `maxbtc-neutron-core` contract. The `execute_collect_fee` handler can be triggered by anyone once the collection period has expired.

The issue arises from mixing inputs from different supply notions: it uses the spot total supply while the provided TWAER implicitly reflects a time-weighted average supply (TWAS). Between the block that publishes the current TWAER and the block that executes `execute_collect_fee`, external actors can change the spot supply by issuing deposit or withdrawal requests to the `maxbtc-neutron-core` contract, which instantly mints and burns MaxBTC ,respectively. This creates a divergence between spot supply and TWAS at fee calculation time.

If the spot supply is higher than the TWAS, the protocol will over-mint fees, diluting MaxBTC holders more than intended. If the spot supply is lower than TWAS, the minted fee will be reduced, allowing griefing attacks against the fee recipient.

A malicious operator can repeatedly cycle deposits and withdrawals to inflate the spot supply just before fee collection, extracting excess fees at minimal cost. Since deposit fees and inflation are collected by the operator, this attack can be profitable and repeated.

A more advanced adversary with MEV capabilities can execute a precise exploit. By monitoring the mempool for Tick messages from the maxbtc-neutron-core operator, they can:

1. Detect when more deposits than withdrawals have occurred.
2. Prepend a withdrawal transaction that burns MaxBTC equal to the new deposits, reducing spot supply.
3. Include a `CollectFee` call within the same withdrawal transaction, forcing the fee to be calculated at a lower spot supply.
4. Allow the `Tick` to finalize.
5. Immediately claim the deposits via the maxbtc-neutron-withdrawal-manager.

This reduces the protocol's minted fees and creates profit whenever the underlying asset price is below the TWAER, or costs only the difference plus gas.

Recommendation

We recommend publishing and consuming a time-weighted average supply (TWAS) alongside the TWAER, and passing the TWAS, not the spot supply, as the `total_supply` input to `calculate_fee_to_mint`. This considerably increases the cost to manipulate the quoted MaxBTC supply and, therefore the minted fees.

Status: Acknowledged

7. Deposit flushing is vulnerable to Denial-of-Service attacks

Severity: Major

When the `maxbtc-neutron-core` contract is in the `Idle` state, and the operator issues the `Tick` message, the `execute_tick_idle` handler in `maxbtc-neutron:contracts/maxbtc-neutron-core/src/contract.rs:209-274` is executed.

In the case where the `maxbtc_burned` value stored in `ACTIVE_BATCH` is non-zero, and the calculated `btc_requested <= deposit_balance`, the `btc_requested` amount is sent to the `maxbtc-neutron-withdrawal-manager` and the batch is set to finalized with the contract remaining in the Idle state.

The remaining deposit amount is not processed until the next `Tick` message is issued by the operator.

A DoS attack is possible by withdrawing a single unit of MaxBTC between `Tick` messages, preventing deposits from progressing further than the `maxbtc-neutron-core` contract. Only a single unit is required because even if `btc_requested` is calculated as zero, the contract remains in the Idle state.

Recommendation

We recommend adding an amount parameter to the `execute_flush_deposits` function and always calling it in `execute_tick_idle` if `btc_requested < deposit_balance`, with the amount set to the difference.

Status: Acknowledged

8. Missing rent-exemption checks for SOL withdrawals

Severity: Major

In `maxbtc-solana:programs/jupiter-helper/src/instructions/withdraw_asset.rs:27-30`, when withdrawing native SOL, the code allows withdrawing the entire lamport balance without ensuring the account maintains rent-exemption.

Withdrawing all lamports could bring the PDA below the rent-exemption threshold, potentially leading to account closure and loss of the PDA address.

Recommendation

We recommend maintaining the minimum rent-exemption balance.

Status: Resolved

9. Sanctioned addresses can purchase and redeem MaxBTC

Severity: Major

The `execute_deposit` handler queries the maxbtc-neutron-allow-list contract in `maxbtc-neutron:contracts/maxbtc-neutron-core/src/contract.rs:470`, which in turn delegates the whitelisting to the zkME KYC service in `maxbtc-neutron:contracts/maxbtc-neutron-allow-list/src/contract.rs:84-93`.

While this check prevents sanctioned addresses from depositing the underlying asset and minting MaxBTC, no such check takes place in the `execute_withdraw` handler in `maxbtc-neutron:contracts/maxbtc-neutron-core`

`/src/contract.rs:521-603`. This allows sanctioned addresses to buy MaxBTC on the open market and then burn it in return for the underlying asset, undermining the intention of adding KYC.

Recommendation

We recommend applying the KYC restriction consistently and validating that the sending address of withdrawal requests is in the allow list.

Status: Resolved

10. Lack of exchange rate liveliness checks allows over-minting of MaxBTC

Severity: Major

The `calculate_mint_amount` function in `maxbtc-neutron:contracts/maxbtc-neutron-core/src/contract.rs:767-794` calculates how much MaxBTC to mint for deposits by dividing the deposit amount by the time-weighted average exchange rate (TWAER), less a deposit fee.

While the TWAER query response contains a `published_at` field, this is not checked to be within reasonable liveness bounds before calculating the mint amount.

This could lead to a situation where the TWAER is stale due to off-chain services failing to issue messages that trigger TWAER publishing and the depositor receives a higher amount of minted MaxBTC than the actual current TWAER would allow.

Recommendation

We recommend adding a configurable liveness parameter and returning an error if the received TWAER is stale by comparing its `published_at` field to the current block timestamp.

Status: Resolved

11. Missing mint alignment check may cause a miscalculated AUM

Severity: Major

In `maxbtc-solana:packages/base/src/formulas.rs:188-209`, the `get_aum` function assumes a fixed positional mapping between custody accounts and their corresponding oracle price feeds.

Specifically, that `remaining_accounts[i]` holds the custody data and `remaining_accounts[5 + i]` holds the corresponding oracle. This hardcoded indexing implicitly assumes that both are passed in the correct order.

However, this relationship is not explicitly validated, and the code does not ensure that the custody and oracle reference the same asset mint.

As a result, misordered inputs could lead to incorrect AUM calculations by pairing mismatched custodies and oracles.

Recommendation

We recommend adding an explicit validation step to ensure that each custody and its paired oracle share the same underlying asset mint.

Status: Resolved

12. Unrestricted access enables sandwich attack on the process handler

Severity: Major

In `maxbtc-solana:programs/jupiter-helper/src/instructions/process.rs:38-47`, the `process_handler` function lacks access restrictions when `allowed_processor` is `None`, allowing any signer to trigger the processing logic.

A malicious actor can monitor for legitimate `process_handler` invocations and strategically front-run or back-run them to perform a sandwich attack.

By manipulating token prices or liquidity immediately before and after the process execution, an attacker can profit from slippage or price impact at the expense of the protocol or its users. The lack of enforced authorization effectively exposes the function to MEV-style exploitation, particularly in volatile or low-liquidity scenarios.

Recommendation

We recommend implementing a predefined slippage limit to mitigate the impact of potential sandwich attacks.

Status: Acknowledged

13. Existing MaxBTC issuer contract migration does not transfer current deposits

Severity: Major

The MaxBTC protocol intends to decouple minting and core logic by migrating from the legacy combined issuer contract to separate maxbtc-neutron-token and maxbtc-neutron-core contracts.

In `maxbtc-neutron:contracts/maxbtc-neutron-token/src/contract.rs:233-421`, the migration logic mirrors the deployment flow from the factory and transfers configuration and state to the new core via the InstantiateMsg.

While the migration handler correctly includes the current deposit balance as part of the instantiation parameters for maxbtc-neutron-core, it fails to actually transfer those funds to the new contract.

As a result, if there are active deposits at the time of migration, the new core will initialize with zero on-chain balance. Any attempt to flush deposits or process withdrawals will fail due to insufficient funds, despite the internal state reflecting a positive deposit balance.

This breaks continuity in deposit tracking and servicing, and will force the client to redeploy or upgrade the maxbtc-neutron-token contract again with additional migration logic to send the missing funds, introducing operational risk and complexity.

Recommendation

We recommend appending a fund transfer message to the migration response that transfers the current deposit balance from the old contract to the newly instantiated maxbtc-neutron-core address. This ensures that on-chain funds remain synchronized with the internal state during migration and avoids post-deployment errors or interruptions in user-facing operations.

Status: Resolved

14. Using the vault account as the Wormhole bridge fee payer allows griefing by fee draining

Severity: Major

The first account passed to the bridge instruction in `maxbtc-solana:programs/wormhole-helper/src/instructions/bridge_tokens.rs:87` is the account that pays for the bridging fee.

However, this is set to the wormhole-helper vault account rather than the signer of the permissionless `BridgeTokens` instruction.

A griefing attacker could drain the lamports allocated to the vault account for fees and rent exemption by repeatedly bridging small token amounts.

Recommendation

We recommend making the `BridgeTokens` instruction signer the Wormhole bridge fee payer.

Status: Resolved

15. Unchecked `unwrap` on optional cap field causes panic

Severity: Major

In `maxbtc-neutron:contracts/maxbtc-neutron-core/src/contract.rs:394`, the owner's `UpdateConfig` handler updates the `deposits_cap` field when provided.

However, the current implementation unconditionally calls `unwrap` on the optional cap value when adding an event attribute, even if that value is explicitly set to `None`.

If the update is intended to clear the cap (i.e., set it to `None`), this will trigger a runtime panic, halting contract execution.

Recommendation

We recommend safely handling the optional case by conditionally logging the updated value only when cap is `Some`.

Status: Resolved

16. Misleading use of `TOTAL_DEPOSITED` leads to protocol lock

Severity: Major

In `maxbtc-neutron:contracts/maxbtc-neutron-core/src/contract.rs:508`, the `TOTAL_DEPOSITED` variable is incremented with each new deposit but never decremented when funds are transferred to the withdrawal manager or moved via CEFFU.

This variable functions as a lifetime cumulative counter, yet it is directly compared to `deposits_cap` when evaluating whether a new deposit should be accepted. As a result, the cap check is based on gross historical inflows rather than the actual assets currently under management (AUM).

This causes the effective cap to drift upward and become overly restrictive, leading the protocol to reject valid deposits even when the actual AUM is well below the intended threshold.

Recommendation

We recommend tracking net deposits in order to enforce the desired cap.

Status: Resolved

17. No validation for lock amount allows negative and zero values

Severity: Minor

In `aum-oracle:contracts/waitasaurus/src/contract.rs:83-104`, The `execute_lock` function accepts a `SignedDecimal256` amount parameter without enforcing that the value is strictly positive. The unlock path in lines `131-133` only rejects when the oracle-reported amount is strictly less than the locked threshold.

For zero or negative lock amounts, this comparison behaves unexpectedly:

- With a zero lock amount: Any non-negative oracle amount passes the check
- With a negative lock amount: Any oracle amount (even zero) passes the check

This allows an authorized locker or owner to set a zero or negative lock amount and immediately unlock regardless of actual asset holdings, defeating the intended gating mechanism.

Recommendation

We recommend adding validation in the `execute_lock` function to ensure amounts are strictly positive. Consider using `Uint256` instead of `SignedDecimal256` to enforce positive values at the type level.

Status: Resolved

18. Fixed Wormhole Helper configuration cannot be updated to match Wormhole fee changes

Severity: Minor

The `update_config_handler` in `maxbtc-solana:programs/wormhole-helper/src/instructions/update_config.rs:13-25` only allows the `Config::target_address` field to be updated.

If the required bridge fee rate changes or the required nonce, then the program will need to be upgraded in order to adapt to the change.

Recommendation

Allow the owner account to also update the `Config::fee` and `Config::nonce` fields.

Status: Resolved

19. Configuration changes while locked can prevent unlocking

Severity: Minor

In `aum-oracle:contracts/waitasaurus/src/contract.rs:56-81`, the `execute_update_config` function allows the owner to change configuration parameters, including the external contract address and asset name, while the contract is in a locked state.

When unlocking, the contract uses the current configuration values rather than the values that were in place when the lock was created. This means changing the contract address or asset name can make it impossible to unlock even if the original conditions are met.

Recommendation

We recommend preventing updates to the `contract` and `asset` fields while the contract is in a locked state.

Status: Resolved

20. Potential division by zero in liquidity calculations

Severity: Minor

In `maxtbc-solana:programs/jupiter-helper/src/instructions/process.rs:175` and `308`, the code performs division operations using `total_supply_jlp` without checking if it is zero.

While JLP tokens should have a non-zero supply in normal operation, the lack of validation could cause a program panic and denial of service.

If the JLP token supply becomes zero (unlikely but theoretically possible), any calls to the `process` instruction would panic, effectively disabling liquidity operations and locking user funds.

Recommendation

We recommend adding a validation check before performing division.

Status: Resolved

21. Liquidity operation fees can be set up to 100%

Severity: Minor

In `maxbtc-solana:programs/jupiter-helper/src/instructions/initialize.rs`, the `maximum_fee_bps` parameter is validated to not exceed 10,000 basis points (i.e., 100%).

However, allowing a full 100% fee introduces a critical behavioral edge case: when `maximum_fee_bps` is set to 10,000, the computed multiplier in fee calculations becomes zero, resulting in `min_assets_amount_out` or `min_jlp_amount_out` being zero as well.

This permits the Jupiter Perpetuals integration to return zero output tokens during liquidity operations while consuming the user's input entirely, effectively enabling 100% of user funds to be extracted under the guise of fees.

Recommendation

We recommend implementing a reasonable upper limit for fees to prevent potential abuse while still allowing flexibility for legitimate fee structures.

Status: Acknowledged

22. Missing validation on collection period allows invalid or inconsistent configuration

Severity: Minor

In `maxbtc-neutron:contracts/maxbtc-neutron-fee-collector/src/contract.rs`, the `collection_period_seconds` parameter is accepted during both initialization and configuration updates without proper validation.

During instantiation, the field is set directly from `msg.collection_period_seconds` without enforcing that the value is positive or meets a minimum threshold. Similarly, in the config update logic, `collection_period_hours` is converted to seconds and assigned without validating that it is greater than zero.

This lack of validation introduces risks of misconfiguration, such as setting the collection period to zero or a value shorter than intended, which may lead to unexpected behavior, excessive fee collection frequency, or unnecessary gas consumption. Notably, the logic assumes a minimum one-hour interval for fee collection, which is not enforced during instantiation.

Recommendation

We recommend validating the collection period before storing it in the contract.

Status: Resolved

23. Unverified trust assumption in Jupiter Perpetuals integration

Severity: Minor

In `maxbtc-solana:programs/jupiter-helper/src/instructions/process.rs:353-355`, the contract performs cross-program invocations (CPI) to the Jupiter Perpetuals program, assuming it will enforce proper validation of account ownership, oracle correctness, and custody consistency.

The contract also passes `remaining_accounts`, including custody and oracle accounts, without validating their identities. It relies entirely on Jupiter Perpetuals to enforce correctness. If the Jupiter program fails to validate these accounts, changes its behavior, or introduces regressions in access control or account routing, it could result in unauthorized fund access, manipulation of oracle input, or incorrect liquidity outcomes.

This reliance on opaque and potentially mutable external validation logic creates a centralization and integrity risk for the protocol's execution flow and fund safety.

Recommendation

We recommend adding explicit account ownership validation before CPI calls where stack size permits. Additionally, write comprehensive integration tests to verify Jupiter Perpetuals' validation behavior and monitor Jupiter Perpetuals for any updates or changes.

Status: Acknowledged

24. Permissionless bridge function allows griefing attacks

Severity: Minor

In `maxbtc-solana:programs/wormhole-helper/src/instructions/bridge_tokens.rs:29-57`, the `bridge_tokens` function is permissionless and can be called by anyone.

While the function correctly bridges tokens from the vault, there is no minimum amount enforcement. This allows malicious actors to repeatedly call this function with small or zero amounts, causing spam and wasting resources.

Recommendation

We recommend adding a minimum amount check and ensuring the bridge amount is greater than zero before proceeding with the bridge operation.

Status: Resolved

25. Owner can withdraw assets regardless of the lock state

Severity: Minor

In `maxbtc-solana:programs/waitosaur/src/instructions/withdraw_asset.rs`, the `withdraw_asset` function does not check the current `WaitosaurState` before allowing withdrawals.

This allows the owner to bypass the entire locking mechanism by withdrawing funds at any time, whether the contract is locked or unlocked, undermining the security guarantees of the waitosaur pattern.

Recommendation

We recommend adding a state check to ensure withdrawals are only allowed when the waitosaur is in the `Unlocked` state, or implement a separate emergency withdrawal mechanism with appropriate safeguards.

Status: Resolved

26. Owner can bypass lock state using `reset_locker`

Severity: Minor

In `maxbtc-solana:programs/waitosaur/src/instructions/reset_locker.rs`, the `reset_locker` function allows the owner to arbitrarily change the contract state from locked to unlocked without any validation or checks.

While this may be intended for emergency scenarios (e.g., backend errors with incorrect lock amounts, market losses preventing meeting lock requirements), it completely undermines the locking mechanism as the owner can force the contract into any state at will, bypassing the intended lock/unlock flow and security guarantees.

Recommendation

We recommend either removing the `reset_locker` function entirely or implementing strict safeguards such as time delays.

Status: Acknowledged

27. Core contract reassignment without namespacing risks withdrawal state corruption

Severity: Minor

In `maxbtc-neutron:contracts/maxbtc-neutron-withdrawal-manager/src/contract.rs:198-229`, the `execute_update_config` function allows the owner to update the core and token contract addresses.

However, the withdrawal manager tracks paid claim amounts using only the `batch_id` as the key, without any namespacing tied to the core contract.

If the owner reassigns the contract to a new core instance that reuses previously used `batch_ids`, the contract may incorrectly associate the new batch with stale accounting data.

This misassociation can result in the contract believing that withdrawals have already been processed, causing legitimate claims to be blocked, underflow errors to occur, or payouts to be incorrectly denied.

Recommendation

We recommend allowing updates to the `core_contract` only during controlled contract migration.

Status: Resolved

28. Protocol centralization due to off-chain operator dependency

Severity: Minor

The protocol's finite state machine (FSM) progression is entirely dependent on actions initiated by a centralized operator. Only this operator can trigger transitions between protocol states, making them the sole actor capable of advancing the system.

Furthermore, many state transitions rely on an external off-chain component, which is implicitly trusted to operate correctly and securely. This architectural design introduces a centralization risk, as any failure, compromise, or unavailability of the operator or backend system can halt protocol operations.

Recommendation

We recommend reducing reliance on a centralized operator, enhancing protocol resilience, and aligning with trust-minimized principles.

Status: Acknowledged

29. Fragile local calculation of output amount depends on external program logic

Severity: Minor

The `jlp_amount_out` function in `maxbtc-solana:programs/jupiter-helper/src/instructions/process.rs:296-328` performs a local computation of expected JLP tokens based on AUM, virtual price, and custody price data.

This approach is fragile, as it replicates logic that is owned and may be modified by the Jupiter Perpetuals program. Any updates or internal changes to how output is computed could desynchronize this helper logic, leading to incorrect results and potential downstream inconsistencies in liquidity provisioning.

While this doesn't directly expose funds to loss, it introduces a latent correctness risk. Operators must now track and align with Jupiter's closed-source logic, requiring updates to this helper whenever the upstream implementation changes, a brittle and error-prone dependency.

Recommendation

We recommend replacing the local calculation with an on-chain invocation of the Jupiter Perpetuals program's `getAddLiquidityAmountAndFee2` instruction.

While integration testing is necessary due to the closed-source nature of Jupiter's implementation, this ensures consistency with the source of truth and eliminates risks stemming from logic duplication.

Status: Resolved

30. Conflicting configuration flags allow ambiguous processor update

Severity: Minor

The `update_config` function in `maxbtc-solana:programs/jupiter-helper/src/instructions/update_conf` `ig.rs:34-49` permits simultaneous setting and removal of the `allowed_processor` configuration field.

Specifically, if both `allowed_processor` is `Some(pubkey)` and `remove_allowed_processor` is `true`, the function executes both branches, leading to unclear or unintended final state.

This ambiguity can confuse downstream logic and auditing by allowing contradictory intent in a single transaction.

Recommendation

We recommend enforcing mutual exclusivity between the `allowed_processor` and `remove_allowed_processor` parameters. If both are set in the same call, the function should return a `CustomError::ConflictingProcessorFlags` or similar.

Status: Resolved

31. Zero-value lock permitted due to missing aggregate amount validation

Severity: Minor

The `lock_handler` function in `maxbtc-solana:programs/waitosaur/src/instructions/lock.rs:14-36` allows the contract to enter a `Locked` state even when all asset values provided in `LockParams` are zero.

This introduces ambiguity in contract behavior and creates potential misuse scenarios where the system appears locked despite no actual funds being committed. It also undermines the semantic meaning of a “lock” operation, which should reflect a meaningful asset constraint.

Recommendation

We recommend enforcing a check to ensure that the aggregate sum of all assets (`usdc`, `wbtc`, `weth`, `wsol`) in the `LockParams` is strictly greater than zero before transitioning to the `Locked` state.

Status: Resolved

32. Single-block instant withdrawal can stifle AUM growth

Severity: Minor

In `maxbtc-neutron:contracts/maxbtc-neutron-withdrawal-manager/src/con` tract.rs, the execute_claim handler allows users to redeem batch tokens as soon as the batch is marked finalized.

During a Tick operation on the maxbtc-neutron-core contract, incoming deposits are first used to satisfy outstanding withdrawals. If funds are sufficient, the withdrawals are marked as finalized and transferred to the withdrawal manager in the same transaction.

A MEV-capable actor can monitor the mempool for an operator's Tick message and front-run it by submitting a withdrawal request sized precisely to fit the difference between the current deposit balance and the total maxbtc_requested. The attacker receives batch redemption tokens in the current batch, which, upon execution of the Tick, will be instantly finalized and covered, making them immediately claimable via execute_claim.

This strategy becomes profitable if the spot price of the MaxBTC backing asset is lower than the prevailing time-weighted average exchange rate (TWAER), allowing the attacker to buy MaxBTC cheaply, withdraw at a higher effective rate, and potentially repay any borrowed capital. The attack can be repeated systematically, enabling front-runners to extract value from user deposits before they are counted toward AUM.

This exploit distorts economic incentives by redirecting all user deposits toward satisfying arbitraged withdrawals, suppressing net AUM growth and reducing capital efficiency for honest participants.

Recommendation

We recommend disabling single-block instant withdrawals by recording the batch finalization block height and requiring that claims may only be completed at least one block after. While this does not eliminate the covered withdrawals arbitrage, it prevents sophisticated single-block executions that may exacerbate any negative effects.

Status: Acknowledged

33. Manual allow list growth can cause deposit Denial-of-Service

Severity: Minor

The maxbtc-neutron-allow-list contract stores a vector of addresses in a single ALLOW_LIST storage item defined in maxbtc-neutron:contracts/maxbtc-neutron-allow-list/src/state.rs:6.

The IsAddressAllowed query handler loads, deserializes and iterates over the entire vector in the worst case, before delegating the whitelist query to zkME.

As there are no size bounds on the manual allow list, an admin could inadvertently set a list large enough to cause out-of-gas errors for contracts that use the query. The maxbtc-neutron-core contract uses the IsAddressAllowed query in the execute_deposits handler in maxbtc-neutron:contracts/maxbtc-neutron-core/src/contract.rs:470 to check whether an address is allowed to deposit the underlying token to mint MaxBTC. An out-of-gas error would block deposits until the allow list size was reduced by the admin.

Additionally, elements in the vector should be deduplicated.

Recommendation

We recommend using a `Map` storage type for `ALLOW_LIST` and checking if a queried address is present as a key to determine allow list inclusion. This reduces the storage read and deserialization costs while turning the cost of inclusion checking from `O(N)` to `O(1)`.

Status: Resolved

34. Lack of validation for KYC service addresses allows for deposit Denial-of-Service

Severity: Minor

In the `maxbtc-neutron-allow-list` the `ZK_ME_SETTINGS` are set by the admin by issuing the `UpdateZkMeSettings` message. If present, those settings are used to query the zkME KYC service contracts in the `IsAddressAllowed` query handler to delegate address whitelisting.

The `UpdateZkMeSettings` message handler does not verify the given addresses are valid before committing them to storage. Invalid addresses will result in an error that will DoS deposits to the `maxbtc-neutron-core` contract. While the address fields are assigned the `Addr` type in `maxbtc-neutron:contracts/maxbtc-neutron-allow-list/src/msg.rs:22-23`, no validation is performed during deserialization.

Recommendation

We recommend always using the `String` type for address fields in untrusted inputs and explicitly obtaining the validated `Addr` using the `Api::addr_validate` function.

Status: Resolved

35. Lack of deposit cost validation can lead to a subtraction overflow

Severity: Minor

The `maxbtc-neutron-core` contract has a `deposit_cost` configuration option in the form of a fee rate, the complement of which is applied in the `calculate_mint_amount` function in `maxbtc-neutron:contracts/maxbtc-neutron-core/src/contract.rs:765-794`.

The `deposit_cost` field is set in the `instantiate` handler in line `55` and in the `execute_update_config` handler in lines `398-401`. No validation checks that the value is less than one are performed in either location.

If the value is inadvertently set to be greater than one, the contract will panic with a subtraction overflow in line `787`. If the value is set to exactly one, the calculated mint amount will be zero and the `execute_deposit` handler will return an error in lines `486-488`. In both cases, there will be a denial of service for depositors.

Recommendation

We recommend adding validation logic in the `instantiate` and `execute_update_config` handlers to check that the proposed `deposit_cost` is less than one.

Status: Resolved

36. Lack of pagination for finalized batches query may lead to out-of-gas error

Severity: Minor

The `FinalizedBatches` query request defined in `maxbtc-neutron:packages/base/src/msg/core.rs:96` contains an optional `batch_id` argument.

If the `batch_id` argument is set, the query handler in `maxbtc-neutron:contracts/maxbtc-neutron-core/src/contract.rs:689-704` either attempts to return the single corresponding batch.

Otherwise, it attempts to load, deserialize and return all of the finalized batches.

As the number of batches increases, any contracts depending on the non-specific batch behaviour of the query may face out-of-gas errors.

Recommendation

We recommend splitting the query into one for a specific batch and another paginated query to request a set of batches.

Status: Resolved

37. Contracts allow for downgrading without changing the contract version

Severity: Minor

The `migration` handler in the maxbtc-neutron-factory, maxbtc-neutron-waitosaur-holder, maxbtc-neutron-withdrawal-manager, maxbtc-neutron-fee-collector, maxbtc-neutron-core, maxbtc-neutron-token, and maxbtc-neutron-exchange-rate-provider contracts does not return an error if the `CONTRACT_VERSION` is lower than the stored contract version, allowing for downgrading. If a downgrade occurs, then the stored contract version is not updated.

This could result in a mismatched state if the downgraded contract tries to use storage removed in a previous upgrade. If the contract is then upgraded, the stored contract version will only be updated if the upgrade version is higher than the original stored version. It will confuse readers of the stored contract version as this version will be incorrect if a downgrade occurs.

Recommendation

We recommend clarifying the desired behavior. If it is required to allow downgrades, then also update the store version. Otherwise, return an error if `storage_version >= version`.

Status: Acknowledged

38. Contracts instantiated by the maxbtc-neutron-factory contract cannot be upgraded

Severity: Minor

The `maxbtc-neutron-factory` contract instantiates all of the other contracts in the repository as part of its own `instantiate` handler in `maxbtc-neutron:contracts/maxbtc-neutron-factory/src/contract.rs:29-294`. It sets itself as the upgrade admin for each of the contracts it instantiates.

However, it has no facility to forward migration messages to the contracts for which it is the upgrade admin. Unless the upgrade admin of the `maxbtc-neutron-factory` contract first upgrades it to a version that has this facility or performs the migration within the factory `migrate` handler, none of the contracts can be updated.

This assumes an upgrade admin is set for this contract and it is not made immutable, in which case all of the contracts it instantiates are also immutable.

Recommendation

We recommend either nominating an upgrade admin in the `InstantiateMsg` or allowing the owner to issue migration messages, which are forwarded to the target contract by the `maxbtc-neutron-factory` contract.

Status: Acknowledged

39. Missing pubkey validation in Wormhole Helper initialization enables misconfiguration

Severity: Minor

The `initialize_handler` function in `maxbtc-solana:programs/wormhole-helper/src/instructions/initialize.rs:17-22` lacks validation checks for `Pubkey` fields provided via `InitializeParams`.

Specifically, the `owner`, `wormhole_token_bridge`, `wormhole_wrapped_metadata`, and `mint` values are accepted without verifying that they are valid, non-default public keys.

This omission opens the door to unintentional misconfiguration, which could result in incorrect routing of assets or assignment of ownership to an unusable key.

Recommendation

We recommend validating each of the fields to ensure they are not set to `Pubkey::default`.

Status: Resolved

40. Missing pubkey validation in Waitosaur initialization enables misconfiguration

Severity: Minor

In `maxbtc-solana:programs/waitosaur/src/instructions/initialize.rs:16-35`, the `initialize_handler` function checks that the `mint` keys are not set to `Pubkey::default`, but it omits similar validation for other configuration parameters.

The `locker`, `unlocker`, `owner`, and each entry in the `recipients` array are accepted without validation, potentially allowing the contract to be initialized with default or invalid public keys.

This increases the risk of misconfigured access controls or broken recipient logic.

Recommendation

We recommend validating each of the fields to ensure they are not set to `Pubkey::default`.

Status: Resolved

41. Configuration updates allowed while the Waitosaur is locked may lead to an inconsistent state

Severity: Minor

Both `maxbtc-solana:programs/waitosaur/src/instructions/update_config.rs` and `maxbtc-neutron:contracts/maxbtc-neutron-waitosaur-holder/src/contract.rs` allow contract configuration to be updated, such as recipient addresses, locker/unlocker roles, asset details, or withdrawal manager, without verifying the system's current state.

This opens the possibility of modifying configuration while the contract is in the `Locked` state, potentially introducing inconsistencies or misrouted funds during a sensitive operational period.

Updating recipients or operational parameters mid-lock may conflict with the assumptions of ongoing processes tied to a specific configuration.

Recommendation

We recommend enforcing a check that the contract is in the `Unlocked` state before permitting any configuration updates.

Status: Resolved

42. Unvalidated recipient address in claim execution may lead to misrouted funds

Severity: Minor

In `maxbtc-neutron:contracts/maxbtc-neutron-withdrawal-manager/src/contract.rs:168`, the `execute_claim` function constructs a `CosmosMsg::Bank::Send` using the provided `recipient` address without performing any validation.

Using an unchecked `recipient` introduces the risk of misrouting funds to an invalid or unintended address, particularly if the input was malformed or manipulated.

Recommendation

We recommend validating the `recipient` address before constructing the send message.

Status: Resolved

43. Inaccessible `SetTokenMetadata` entrypoint leaves token without metadata

Severity: Minor

The `execute_set_token_metadata` function in `maxbtc-neutron:contracts/maxbtc-neutron-token/src/contract.rs:181-210` is protected by an ownership check via `cw_ownable::assert_owner`, requiring the message sender to match the contract owner.

However, due to the deployment architecture, the token contract is instantiated with the maxbtc-neutron-core contract as its owner, while that core contract does not expose any execute message that would forward a `SetTokenMetadata` call.

As a result, after deployment, no on-chain actor has the ability to invoke this function, rendering the metadata-setting logic effectively unreachable.

Moreover, although the factory assigns itself as the wasm admin, this role permits migration but does not override cw_ownable ownership logic, meaning the metadata cannot be updated post-deploy unless explicitly configured during instantiation or migration.

Recommendation

We recommend setting the token metadata at the time of token instantiation or during a controlled migration.

Status: Acknowledged

44. Inefficient ZkMe error handling

Severity: Informational

In `maxbtc-neutron:contracts/maxbtc-neutron-allow-list/src/contract.rs:84-95`, when the ZkMe contract query fails, the function returns `false` instead of propagating the error.

This silent failure makes it difficult to distinguish between a user not being approved and the ZkMe service being unavailable, potentially denying access to legitimate users during ZkMe outages.

Recommendation

We recommend propagating ZkMe query errors or implementing a fallback mechanism that allows the protocol to continue operating when ZkMe is unavailable, such as falling back to the allow list only.

Status: Acknowledged

45. Configurable discriminator could lead to incorrect message types

Severity: Informational

In `maxbtc-solana:programs/wormhole-helper/src/config.rs:5`, the discriminator field is configurable.

The discriminator determines the message type for Wormhole, and using an incorrect value could lead to message parsing errors on the destination chain.

Recommendation

We recommend hardcoding the discriminator value to prevent configuration errors.

Status: Resolved

46. Broad version range in migration logic may lead to unintended upgrades

Severity: Informational

In `maxbtc-neutron:contracts/maxbtc-neutron-token/src/contract.rs:239`, the migration logic migrates the contract using the `cw2::set_contract_version` whenever `storage_version < version`.

This permissive check allows migration from any earlier version, even if intermediary steps or structural changes are required. Given that this migration is tailored to a specific release, allowing broad version transitions could result in missed invariants, broken state assumptions, or partial upgrades if invoked from an unsupported version.

Recommendation

We recommend restricting the migration to explicitly supported version pairs.

Status: Acknowledged

47. Missing event emissions reduce observability

Severity: Informational

In `maxbtc-neutron:contracts/maxbtc-neutron-waitosaur-holder/src/contract.rs`, the contract functions only emit basic attributes without detailed information about locked amounts, timestamps, or transferred values. This reduces observability and makes it difficult to monitor contract operations off-chain.

Recommendation

We recommend adding comprehensive event attributes for all state changes, including lock amounts, timestamps, unlock transfer amounts, and configuration changes.

Status: Resolved

48. Contracts should implement a two-step ownership transfer

Severity: Informational

The maxbtc-neutron-fee-collector allows 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.

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

Status: Resolved

49. Incomplete TODO comments

Severity: Informational

In `maxbtc-neutron:maxbtc-neutron-core/src/contract.rs:298`, `308`, there are TODO comments indicating incomplete implementation for `deposit_pending` and `deposit_jlp` tick functions. These functions currently only perform state transitions without implementing the full logic described in the documentation.

Recommendation

We recommend completing the implementation of these functions according to the architectural specification before deployment.

Status: Resolved

50. Unused error variants increase codebase complexity

Severity: Informational

In `maxbtc-neutron:contracts/maxbtc-neutron-waitosaur-holder/src/error.rs`, the `ContractError` enum defines several error variants such as `NoDataInContract` and `NoAssetFound` that are currently unused in the contract logic.

Maintaining unused error types introduces unnecessary noise and increases the cognitive load for developers reviewing the error surface, especially in security-sensitive contracts where each error may imply a defensive check.

Recommendation

We recommend removing unused error variants.

Status: Resolved

51. Incorrect event emission on token minting

Severity: Informational

In `maxbtc-neutron:contracts/maxbtc-neutron-token/src/contract.rs:133-154`, the `execute_mint` function allows the contract owner to mint tokens by constructing a token factory mint message.

While the emitted event logs the configured `cfg.denom`, the function does not explicitly validate that the minted `amount.denom` matches `cfg.denom`.

Consequently, the minting of redemption tokens is misleadingly tagging the event with an incorrect denom.

Recommendation

We recommend describing the correct denom in the emitted event.

Status: Resolved

52. Missing associated token account mint constraint

Severity: Informational

In `maxbtc-solana:programs/wormhole-helper/src/instructions/withdraw_asset.rs:81`, there is no `associated_token::mint = mint` constraint for the `recipient_ata` account.

While this results in a `MintMismatch` error from the spl_token program rather than a loss of funds, it is more efficient to add an explicit constraint and fail early.

Recommendation

We recommend adding the `associated_token::mint = mint` constraint for the `recipient_ata` account.

Status: Resolved

53. Use of magic numbers increases maintenance costs

Severity: Informational

There are a number of instances where magic numbers are used:

- `maxbtc-solana:programs/jupiter-helper/src/instructions/update_config.rs:21`
- `maxbtc-solana:programs/jupiter-helper/src/instructions/initialize.rs:22`
- `maxbtc-solana:programs/jupiter-helper/src/instructions/process.rs:172`
- `maxbtc-solana:programs/jupiter-helper/src/instructions/process.rs:197`

- `maxbtc-solana:programs/jupiter-helper/src/instructions/process.rs:305`

The use of magic numbers decreases readability and increases maintenance costs.

Recommendation

We recommend replacing the use of magic numbers with named constants.

Status: Resolved

54. Misleading error

Severity: Informational

In `maxbtc-neutron:contracts/maxbtc-neutron-core/src/contract.rs:336`, the `execute_mint_fee` handler returns a `ContractError::InvalidDepositDenom` if the requested coin minted is not the MaxBTC denom.

As the action is not a deposit, the error will result in confusion if returned, increasing the cost of debugging configuration issues.

Recommendation

We recommend returning an `InvalidMintDenom` error variant.

Status: Resolved

55. Dead code and misleading comments increase maintenance costs

Severity: Informational

There are a number of instances of dead code:

- In `maxbtc-solana:packages/base/src/formulas.rs:157-161`, the calls to `Decimal::floor` and `Decimal::ceil` have no effect.
- In `maxbtc-solana:programs/waitosaur/src/instructions/withdraw_asset.rs:100-101`, `init_if_needed` and `payer = signer` are not required as the account must already exist for withdrawals to take place.
- In `maxbtc-solana:programs/wormhole-helper/src/instructions/bridge_tokens.rs:112-118`, the `rent`, `system_program`, `token_program` and

`wormhole_core_bridge` accounts are unused by the Wormhole Token Bridge `TransferWrapped` instruction.

- In maxbtc-solana:programs/wormhole-helper/src/instructions/withdraw_asset.rs:68-69, `init_if_needed` and `payer = signer` are not required as the account must already exist for withdrawals to take place.

As well as a number of misleading comments:

- In `maxbtc-neutron:contracts/maxbtc-neutron-core/src/contract.rs:607-608`, the referenced `deposit_flush_period` configuration option does not exist and the `execute_flush_deposits` function cannot be triggered externally by any account, it is only called by `execute_tick_idle`.
- In `maxbtc-neutron:contracts/maxbtc-neutron-token/src/contract.rs:143`, the `execute_mint` function does not always mint MaxBTC, it also mints withdrawal batch redemption tokens.
- In `maxbtc-neutron:contracts/maxbtc-neutron-token/src/contract.rs:170-171`, the action is burning not minting and it is not always MaxBTC being burned, the function also burns redemption tokens. Additionally the binding is given the name `mint_msg` when it should be `burn_msg`.
- In `maxbtc-neutron:contracts/maxbtc-neutron-factory/src/contract.rs:73`, the `canonical_creator` is not the core contract, it is the `maxbtc-neutron-factory` contract. This is repeated in every subsequent `instantiate2_address` call.
- In `maxbtc-neutron:contracts/maxbtc-neutron-factory/src/contract.rs:143`, the `admin` is not the core contract, it is the `maxbtc-neutron-factory` contract. This is repeated in every subsequent construction of `WasmMsg::Instantiate2`.
- In `maxbtc-solana:programs/wormhole-helper/src/instructions/initialize.rs:78`, the program is the `wormhole-helper` not the `jupiter-helper`.
- In `maxbtc-solana:programs/waitosaur/src/instructions/unlock.rs:154`, none of the accounts specified below the comment are being initialized.

Both decrease the readability and increase maintenance costs.

Recommendation

We recommend removing the instances of dead code and correcting the misleading comments enumerated above.

Status: Resolved