CoinFabrik

# Security Audit Report

## AIBTC - DAO

June 2025

V202411

# Executive Summary

**CoinFabrik** was asked to audit the contracts for **AIBTC's DAO** project.

**During this audit we found one critical issue, four high issues, four medium issues and several minor issues. Also, several enhancements were proposed.**

Eleven issues were resolved and two were acknowledged. Two enhancements were implemented.

# Scope

The audited files are from the git repository located at https://github.com/aibtcdev/aibtcdev-daos/, in the `./contracts/` directory. The audit is based on the commit `083744564397575d1b0f580cb300fb14d01fe11a`. Fixes were checked on commit `470cca7903b4c3cf408c0aa6bc5f8683faa54732`.

The scope for this audit includes and is limited to the following files:

- `./agent/aibtc-agent-account.clar`: Two-party account for agent and owner.
- `./core/aibtc-dao-run-cost.clar`: Multi-signature wallet designed to hold and manage operational fees.
- `./dao/aibtc-base-dao.clar`: Core executor for the DAO.
- `./dao/actions/aibtc-action-send-message.clar`: Provides a standardized method for the DAO to send a string message using a dedicated on-chain messaging system.
- `./dao/extensions/aibtc-action-proposal-voting.clar`: Extension that manages the entire lifecycle of governance proposals.
- `./dao/extensions/aibtc-dao-charter.clar`: Extension that serves as an on-chain, version-controlled repository for a DAO's charter. Its primary purpose is to store and manage a string of text representing the DAO's mission, values, and principles.
- `./dao/extensions/aibtc-dao-epoch.clar`: Extension for calculating the current operational epoch for the DAO.
- `./dao/extensions/aibtc-dao-users.clar`: Extension that functions as the user database for the DAO.
- `./dao/extensions/aibtc-onchain-messaging.clar`: Open on-chain messaging service.
- `./dao/extensions/aibtc-rewards-account.clar`: Treasury for DAO proposal rewards.
- `./dao/extensions/aibtc-token-owner.clar`: Extension that provides management functions for the DAO token.

- `./dao/extensions/aibtc-treasury.clar`: Extension that manages DAO fungible tokens.
- `./dao/proposals/aibtc-base-initialize-dao.clar`: Proposal that performs the crucial initial setup for the entire DAO ecosystem.
- `./dao/token/aibtc-faktory-dex.clar`: Exchange and launchpad for DAO token.
- `./dao/token/aibtc-pre-faktory.clar`: Pre-launch token distribution platform.
- `./dao/token/xyk-pool-sbtc-aibtc-v-1-1.clar`: AMM liquidity pool.

No other files in this repository were audited. Its dependencies are assumed to work according to their documentation. Also, no tests were reviewed for this audit.

# Findings

In the following table we summarize the security issues we found in this audit. The severity classification criteria and the status meaning are explained below. This table does not include the enhancements we suggest to implement, which are described in a specific section after the security issues.

Each severity label is detailed in the Severity Classification section. Additionally, the statuses are explained in the Issues Status section.

| Id | Title | Severity | Status |
|---|---|---|---|
| CR-01 | Proposal Replay | Critical | **Resolved** |
| HI-01 | Unvalidated External Contract Address in Agent Account | High | **Resolved** |
| HI-02 | DEX Market Incorrectly Initialized as Open at Deployment | High | **Resolved** |
| HI-03 | DEX Can Be Reopened After Finalization | High | **Resolved** |
| HI-04 | Arithmetic Underflow Causes Denial-of-Service in Presale | High | **Resolved** |
| ME-01 | Confirmation Applied to Mismatched Proposal Parameters | Medium | **Resolved** |
| ME-02 | Incorrect Balance Check for Proposal Creation Fee | Medium | **Resolved** |
| ME-03 | Single-Attempt Execution Prevents Recovery from Transient Failures | Medium | **Acknowledged** |

| Id | Title | Severity | Status |
|---|---|---|---|
| ME-04 | Lack of Initialization Guard Allows Pool Reset | Medium | **Acknowledged** |
| LO-01 | Execution Overwrites Proposal Creation Timestamp | Low | **Resolved** |
| LO-02 | Redundant Data Storage | Low | **Resolved** |
| LO-03 | User createdAt Timestamp is Inaccurate | Low | **Resolved** |
| LO-04 | Lack of Access Control Allows Spam | Low | **Resolved** |

# Critical Severity Issues

## CR-01 Proposal Replay

### Location

- `./core/aibtc-dao-run-cost.clar`

### Classification

- CWE-863: Incorrect Authorization[1]

### Description

The `SetAssetProposals` and `TransferProposals` maps do not include an `executed` flag. This allows a proposal to be executed multiple times before the proposal's expiration window closes. This could be abused to drain funds by re-executing transfer proposals or to maliciously toggle asset statuses.

### Recommendation

Add an `executed` field to the map structures. Within the `execute-set-asset` and `execute-transfer` functions, add a check before execution and set the flag to true upon execution.

### Status

**Resolved**. Fixed according to recommendation.

[1] https://cwe.mitre.org/data/definitions/863.html

# High Severity Issues

## HI-01 Unvalidated External Contract Address in Agent Account

### Location

- `./agent/aibtc-agent-account.clar: 176, 198, 219, 238`

### Classification

- CWE-20: Improper Input Validation[2]

### Description

The functions for DAO interaction (`create-action-proposal`, `vote-on-action-proposal`, etc.) accept a `voting-contract` address as a parameter but do not validate it against a pre-approved list. An authorized user (owner or agent) could be tricked by a phishing attack into providing a malicious contract address, causing the agent account to interact with an untrusted contract.

### Recommendation

Implement a whitelist mechanism and check against it.

### Status

**Resolved**. Fixed according to the recommendation.

## HI-02 DEX Market Incorrectly Initialized as Open at Deployment

### Location

- `./dao/token/aibtc-faktory-dex.clar: 286`

### Classification

- CWE-665: Improper Initialization[3]

### Description

The contract's initializer block at deployment explicitly sets `(var-set open true)`. This directly contradicts the intended control flow, where the market should only be opened via a call to the

[2] https://cwe.mitre.org/data/definitions/20.html
[3] https://cwe.mitre.org/data/definitions/665.html

`open-market` function, which is gated by a check calling the `aibtc-pre-faktory` contract. This premature opening bypasses the intended multi-contract launch sequence.

### Recommendation

The `(var-set open true)` line should be removed from the deployment block. The contract should be initialized with `(define-data-var open bool false)`.

### Status

**Resolved**. Fixed according to recommendation.

## HI-03 DEX Can Be Reopened After Finalization

### Location

- `./dao/token/aibtc-faktory-dex.clar`

### Description

The `buy` function correctly sets `(var-set open false)` upon reaching its goal, but the `open-market` function's only check is against the `aibtc-pre-faktory` contract. If that contract's `is-market-open` flag is not disabled post-finalization, anyone can call `open-market`, resetting the balances and reopening the DEX. This would allow users to trade against a "zombie" pool whose primary liquidity has already been migrated, likely leading to a loss of funds.

### Recommendation

The DEX finalization process must be truly final. Introduce a new state variable, e.g., `(define-data-var finalized bool false)`. In the `buy` function, when the `TARGET_STX` is reached, this variable should be set to `true`. The `open-market` function must then be updated to also check this flag: `(asserts! (not (var-get finalized)) ERR-MARKET-CLOSED)`.

### Status

**Resolved**. Fixed according to recommendation.

## HI-04 Arithmetic Underflow Causes Denial-of-Service in Presale

### Location

- `./dao/token/aibtc-pre-faktory.clar`

### Description

The `get-max-seats-allowed` function contains an unprotected subtraction operation that might lead to a transaction-crashing underflow.

In `(- MIN-USERS (var-get total-users))`, the sale is designed to continue if `MIN-USERS` is reached and there are available seats. As soon as the number of participants (`total-users`) surpasses the `MIN-USERS` constant, this line will underflow.

Because this function is essential for a user to determine how many seats they can buy, its failure will break the dynamic cap logic and could halt participation in the sale.

### Recommendation

Rewrite the function's logic to be defensive against underflow.

### Status

**Resolved**. Fixed according to recommendation.

## Medium Severity Issues

### ME-01 Confirmation Applied to Mismatched Proposal Parameters

### Location

- `./core/aibtc-dao-run-cost.clar`

### Classification

- CWE-20: Improper Input Validation[4]

### Description

The public functions (`set-owner`, `transfer-dao-token`, etc.) combine proposal creation and confirmation, allowing a malicious owner to hijack proposals. This is exploitable via a front-running attack:

1. An attacker (who is also an owner) monitors the mempool for a legitimate proposal transaction for a given nonce.

[4] https://cwe.mitre.org/data/definitions/20.html

2. The attacker sees the transaction and immediately submits their own transaction with the same nonce but with malicious parameters (e.g., sending funds to their own address).
3. By paying a higher fee, the attacker's transaction is mined first, creating the malicious proposal at the target nonce.
4. When the original, legitimate transaction and subsequent confirmations from honest owners are mined, they unknowingly add confirmations to the attacker's malicious proposal.

This breaks the fundamental integrity of the multi-sig, allowing an attacker to execute a proposal that the other owners never intended to approve.

### Recommendation

Modify each public function to first check if a proposal for the nonce exists. If it exists, the function must assert that all parameters in the function call exactly match the stored parameters before proceeding to confirm.

### Status

**Resolved**. Fixed according to recommendation.

## ME-02 Incorrect Balance Check for Proposal Creation Fee

### Location

- `./dao/extensions/aibtc-action-proposal-voting.clar: 189`

### Classification

- CWE-670: Always-Incorrect Control Flow[5]

### Description

In `create-action-proposal`, the contract checks if the user's balance is greater than the sum of `VOTING_BOND` and `AIBTC_DAO_RUN_COST_AMOUNT`. However, the code later reveals that only the `VOTING_BOND` is transferred from the user; the `AIBTC_DAO_RUN_COST_AMOUNT` is paid by the DAO treasury via an `as-contract` call. This creates an unnecessary financial requirement for the user, preventing them from creating a proposal even if they can afford the actual cost they must pay.

[5] https://cwe.mitre.org/data/definitions/670.html

### Recommendation

Remove the `AIBTC_DAO_RUN_COST_AMOUNT` from the initial balance check.

### Status

**Resolved**. Fixed according to recommendation.

## ME-03 Single-Attempt Execution Prevents Recovery from Transient Failures

### Location

- `./dao/extensions/aibtc-action-proposal-voting.clar`

### Description

The `conclude-action-proposal` function attempts to execute the `run` function on the target action contract only once. If this action fails for any reason (e.g., a temporary network condition, or the action needs to be executed in a very specific block), the proposal is marked as concluded, the reward is lost, and there is no mechanism to retry the execution.

### Recommendation

Decouple the proposal conclusion from its execution. One approach is to have `conclude-action-proposal` finalize the vote and mark the proposal as passed and executable. Then, create a new, separate public function, which can be called multiple times within the execution window until the action succeeds.

### Status

**Acknowledged**. The development team stated this is a design decision.

## ME-04 Lack of Initialization Guard Allows Pool Reset

### Location

- `./dao/token/xyk-pool-sbtc-aibtc-v-1-1.clar: 323-354`

### Description

The `create-pool` function can be called multiple times by the `CORE_ADDRESS`. The function sets all the fundamental pool parameters (`x-token`, `y-token`, `name`, `symbol`, etc.) but lacks a check to ensure it only runs once. A subsequent call would completely reset the pool's configuration,

de-registering the original assets and breaking the pool's integrity. This could lead to a loss of funds for existing liquidity providers if, for instance, the pool is re-initialized with different tokens.

### Recommendation

Modify the `create-pool` function to prevent re-initialization.

### Status

**Acknowledged**. Development team stated `CORE_ADDRESS` is assigned to Bitflow core contract and this contract asserts pool can only be created once.

## Low Severity Issues

### LO-01 Execution Overwrites Proposal Creation Timestamp

### Location

- `./core/aibtc-dao-run-cost.clar: 369, 453`

### Classification

- CWE-221: Information Loss or Omission[6]

### Description

In the `execute-set-owner` and `execute-set-confirmations` functions, the created field of a proposal is overwritten with the current block height upon execution. This action erases valuable on-chain audit data, specifically the original creation timestamp of the proposal.

### Recommendation

To preserve the original creation timestamp, use `merge` to update only the `executed` status.

### Status

**Resolved**. Fixed according to recommendation.

[6] https://cwe.mitre.org/data/definitions/221.html

## LO-02 Redundant Data Storage

### Location

- `./dao/extensions/aibtc-dao-charter.clar`

### Description

The contract stores the charter's text in two places: the `daoCharter` data variable (for the current version) and the `CharterVersions` map (for historical versions). While this provides quick access to the current charter, it also introduces data redundancy. Every time `set-dao-charter` is called, the new text is written to both the variable and the map, increasing the transaction's storage cost.

### Recommendation

Remove the `daoCharter` data variable. The `get-current-dao-charter` read-only function can be modified to fetch the charter text directly from the `CharterVersions` map using the `currentVersion` variable: `(map-get? CharterVersions (var-get currentVersion))`. This makes the map the single source of truth for all charter data.

### Status

**Resolved**. Fixed according to recommendation.

## LO-03 User createdAt Timestamp is Inaccurate

### Location

- `./dao/extensions/aibtc-dao-charter.clar`

### Description

The `get-or-create-user-index` function sets a new user's `createdAt` timestamp to `DEPLOYED_BURN_BLOCK`, which is the block height when the contract was deployed, not when the user was actually created. Similarly, the `increase/decrease-user-reputation` functions overwrite the entire `UserData` map entry, resetting the `createdAt` timestamp to this same constant value every time a user's reputation is modified. This results in inaccurate on-chain data.

### Recommendation

When creating a new user, the `createdAt` field should be set to the current block height, `burn-block-height`. When updating a user's reputation, the contract should use merge to modify only the `reputation` field, leaving the original `address` and `createdAt` values intact.

### Status

**Resolved**. Fixed according to recommendation.

## LO-04 Lack of Access Control Allows Spam

### Location

- `./dao/extensions/aibtc-onchain-messaging.clar`

### Description

The `send` function is permissionless, allowing any principal to call it. While this is likely an intentional design choice for a public messaging system, it creates a vector for spam. A malicious actor could repeatedly call the function, creating a high volume of events that could clutter monitoring services or incur costs for off-chain indexers that process these print events.

### Recommendation

This is a design trade-off. To mitigate potential spam, the DAO could consider implementing a fee-based system for non-DAO/non-holder messages.

### Status

**Resolved**. Fixed according to recommendation.

# Enhancements

These items do not represent a security risk. They are best practices that we suggest implementing.

| Id | Title | Status |
|---|---|---|
| EN-01 | Redundant Block Height Check in Conclusion | Not implemented |
| EN-02 | Unnecessary Code for Optional Parameters | **Implemented** |
| EN-03 | Redundant Initialization Call | **Implemented** |

| Id | Title | Status |
| --- | --- | --- |
| EN-04 | Misleading Variable and Event Naming | Not implemented |

## EN-01 Redundant Block Height Check in Conclusion

### Location

- `./dao/extensions/aibtc-action-proposal-voting.clar: 493-495`

### Description

The `conclude-action-proposal` function asserts both `(>= burn-block-height (get voteEnd ...))` and `(>= burn-block-height (get execStart ...))`. Since `execStart` is defined as `voteEnd` plus a delay, the check against `execStart` makes the check against `voteEnd` logically redundant and wastes a small amount of gas.

### Recommendation

Remove the redundant check against `voteEnd`. The single check against `execStart` is sufficient to ensure the function is called in the correct phase.

### Status

**Not implemented**.

## EN-02 Unnecessary Code for Optional Parameters

### Location

- `./dao/extensions/aibtc-action-proposal-voting.clar: 206-209, 234-237, 316-323`

### Description

The contract repeatedly uses the pattern `(if (is-some memo) memo none)` when handling the optional `memo` parameter. This is unnecessary, as assigning the optional parameter directly would have the exact same result.

A similar pattern is used in `vote-on-action-proposal` to get the previous vote record. Directly using `get` on an optional value will return another optional where the asked value is wrapped.

### Recommendation

Simplify the code by removing the redundant `if` statements. Assign the `memo` parameter directly. For retrieving the `voterRecord`'s inner values, use the `get` function for cleaner, more idiomatic code.

### Status

**Implemented**.

## EN-03 Redundant Initialization Call

### Location

- `./dao/proposals/aibtc-base-initialize-dao.clar`

### Description

This contract calls the treasury to whitelist the DAO token. However, the treasury contract already enables this specific token by default in its own deployment code. This `allow-asset` call during initialization is therefore redundant, as it sets a value that is already present in the treasury's `AllowedAssets` map.

### Recommendation

Remove that call from the `execute` function of this initialization contract.

### Status

**Implemented**.

## EN-04 Misleading Variable and Event Naming

### Location

- `./dao/token/aibtc-pre-faktory.clar`

### Description

The variable `stx-balance` and associated event fields are used to track `sBTC` balances, not `STX`. This is confusing for off-chain indexers and anyone auditing the contract, as it misrepresents the asset being handled.

### Recommendation

Rename `stx-balance` to `sbtc-balance` throughout the contract, including in data variables and print event payloads.

### Status

**Not implemented**.

# Other Considerations

The considerations stated in this section are not right or wrong. We do not suggest any action to fix them. But we consider that they may be of interest to other stakeholders of the project, including users of the audited contracts, token holders or project investors.

## Centralization

AIBTC employs a model of governance-controlled centralization, where the main `aibtc-base-dao` contract acts as the central authority, delegating all specific functions to a suite of distinct extension contracts. The security of this model hinges on a critical design choice: the authorization scheme is monolithic. Once an extension is approved by a token-holder vote, it gains total authority to call protected functions across the entire ecosystem, including commanding the treasury or other extensions. This means a single vulnerable extension can compromise the whole system. Therefore, significant trust is placed not only in the initial, developer-defined set of extensions but also in the governance process to perpetually guard against adding flawed extensions and manage the ultimate risk of the token-owner contract, which can transfer away total control of the system.

## Upgrades

The smart contracts do not provide an upgradability mechanism.

# About CoinFabrik

CoinFabrik is a research and development company specialized in Web3, with a strong background in cybersecurity. Founded in 2014, we have worked on over 500 decentralization projects, including EVM-based and other platforms like Solana, Algorand, and Polkadot. Beyond development, we offer security audits through a dedicated in-house team of senior cybersecurity

professionals, working on code in languages such as Substrate, Solidity, Clarity, Rust, TEAL, and Stellar Soroban.

Our team has an academic background in computer science, software engineering, and mathematics, with accomplishments including academic publications, patents turned into products, and conference presentations. We actively research in collaboration with universities worldwide, such as Cornell, UCLA, and École Polytechnique in Paris, and maintain an ongoing collaboration on knowledge transfer and open-source projects with the University of Buenos Aires, Argentina. Our management and people experience team has extensive expertise in the field.

# Methodology

CoinFabrik was provided with the source code, including automated tests that define the expected behavior, and general documentation about the project. Our auditors spent three weeks auditing the source code provided, which includes understanding the context of use, analyzing the boundaries of the expected behavior of each contract and function, understanding the implementation by the development team (including dependencies beyond the scope to be audited) and identifying possible situations in which the code allows the caller to reach a state that exposes some vulnerability. Without being limited to them, the audit process included the following analyses.

- Arithmetic errors
- Race conditions
- Misuse of block timestamps
- Denial of service attacks
- Excessive runtime usage
- Missing or misused function qualifiers
- Needlessly complex code and contract interactions
- Poor or nonexistent error handling
- Insufficient validation of the input parameters
- Incorrect handling of cryptographic signatures
- Centralization and upgradeability

After delivering a report with our findings, the development team had the opportunity to comment on every finding and fix the issues they considered convenient. Once fixed and/or commented, our team ran a second review process to verify that the changes to the code effectively solve the issues found and do not unintentionally add new ones. This report includes the final status after the second review.

# Severity Classification

Security risks are classified as follows[7]:

| | |
|---|---|
| Critical | • Manipulation of governance voting result deviating from voted outcome and resulting in a direct change from intended effect of original results<br>• Direct theft of any user funds, whether at-rest or in-motion, other than unclaimed yield<br>• Direct theft of any user NFTs, whether at-rest or in-motion, other than unclaimed royalties<br>• Permanent freezing of funds<br>• Permanent freezing of NFTs<br>• Unauthorized minting of NFTs<br>• Predictable or manipulable RNG that results in abuse of the principal or NFT<br>• Unintended alteration of what the NFT represents (e.g. token URI, payload, artistic content)<br>• Protocol insolvency |
| High | • Theft of unclaimed yield<br>• Theft of unclaimed royalties<br>• Permanent freezing of unclaimed yield<br>• Permanent freezing of unclaimed royalties<br>• Temporary freezing of funds<br>• Temporary freezing NFTs |

[7] This classification is based on the smart contract Immunefi severity classification system version 2.3. https://immunefi.com/immunefi-vulnerability-severity-classification-system-v2-3/

| | |
|---|---|
| Medium | • Smart contract unable to operate due to lack of token funds<br>• Block stuffing<br>• Griefing (e.g. no profit motive for an attacker, but damage to the users or the protocol)<br>• Theft of gas<br>• Unbounded gas consumption<br>• Security best practices not followed |
| Low | • Contract fails to deliver promised returns, but doesn't lose value<br>• Other security issues with minor impact |

# Issue Status

An issue detected by this audit has one of the following statuses:

- **Unresolved:** The issue has not been resolved.
- **Resolved:** Adjusted program implementation to eliminate the risk.
- **Partially Resolved:** Adjusted program implementation to eliminate part of the risk. The other part remains in the code, but is a result of an intentional decision.
- **Acknowledged:** The issue remains in the code, but is a result of an intentional decision. The reported risk is accepted by the development team.
- **Mitigated:** Implemented actions to minimize the impact or likelihood of the risk.

# Disclaimer

This audit report has been conducted on a **best-effort basis within a tight deadline defined by time and budget constraints**. We reviewed only the specific smart contract code provided by the client at the time of the audit, detailed in the Scope section. We do not review other components that are part of the solution: neither implementation, nor general design, nor business ideas that motivate them.

While we have employed the latest tools, techniques, and methodologies to identify potential vulnerabilities, **this report does not guarantee the absolute security of the contracts, as undiscovered vulnerabilities may still exist.** Our findings and recommendations are

suggestions to enhance security and functionality and are not obligations for the client to implement.

The results of this audit are valid solely for the code and configurations reviewed, and any modifications made after the audit are outside the scope of our responsibility. CoinFabrik disclaims all liability for any damages, losses, or legal consequences resulting from the use or misuse of the smart contracts, including those arising from undiscovered vulnerabilities or changes made to the codebase after the audit.

This report is intended exclusively for the **AIBTC** team and should not be relied upon by any third party without the explicit consent of CoinFabrik. Blockchain technology and smart contracts are inherently experimental and involve significant risk; users and investors should fully understand these risks before deploying or interacting with the audited contracts.

# Changelog

| Date | Description |
|---|---|
| 2025-06-25 | Initial report based on commit `083744564397575d1b0f580cb300fb14d01fe11a`. |
| 2025-07-11 | Reaudit report based on the fixes in commit `470cca7903b4c3cf408c0aa6bc5f8683faa54732`. |