StakingDao's Strategy v3 Audit

May 2024

By CoinFabrik

v202311

Executive Summary

CoinFabrik was asked to audit the contracts for the strategy v3 project for StakingDao.

During this audit we found no critical-severity or high-severity issues. We found 2 medium-severity issues and 2 minor-severity issues. Also, enhancements were proposed.

Scope

The audited files are from the git repository located at https://github.com/StackingDAO/StackingDAO. The audit is based on the commit `edebe00d49cff4c4dc1bb752cfe12f550d7e49ec`. We were also asked to review pull requests https://github.com/StackingDAO/StackingDAO/pull/220/files and https://github.com/StackingDAO/StackingDAO/pull/213/files . Changes were checked over the `f6b1f7f00bdf908a8fefe7eafd55c91dbe10e179` commit.

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

- `clarity/contracts/version-2/strategy-v3.clar`: Stacking strategy version 3.
- `clarity/contracts/version-2/strategy-v3-algo-v1.clar`: Reach-target & lowest combination algorithm implementations.
- `clarity/contracts/version-2/strategy-v3-delegates-v1.clar`: Delegation amount calculations.
- `clarity/contracts/version-2/strategy-v3-pools-v1.clar`: Computes amount of STX to stack per pool.

No other files in this repository were audited; however, some of the dependencies were reviewed and we list some bugs included therein which have impact in the scoped contracts. Also, no tests were reviewed for this audit.

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 a week 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 gas 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

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.

ID	Title	Severity	Status
ME-01	Broad Permissions For Admin Function	Medium	Mitigated.
ME-02	Failed Comparison of Delegates	Medium	Fixed.
MI-01	Unrestricted Setter For a Limited Variable	Minor	Fixed.
MI-02	Coarse Permissions Check Via check-is-protocol()	Minor	Acknowledged.

Severity Classification

Security risks are classified as follows:

- **Critical:** These are issues that we manage to exploit. They compromise the system seriously. Blocking bugs are also included in this category. They must be fixed **immediately**.

- **High:** These refer to a vulnerability that, if exploited, could have a substantial impact, but requires a more extensive setup or effort compared to critical issues. These pose a significant risk and **demand immediate attention**.

- **Medium:** These are potentially exploitable issues. Even though we did not manage to exploit them or their impact is not clear, they might represent a security risk in the near future. We suggest fixing them **as soon as possible**.
- **Minor:** These issues represent problems that are relatively small or difficult to take advantage of, but might be exploited in combination with other issues. These kinds of issues do not block deployments in production environments. They should be taken into account and be fixed **when possible**.

Issues Status

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

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

Critical-Severity Issues

No issues found.

High-Severity Issues

No issues found.

Medium-Severity Issues

ME-01 Broad Permissions For Admin Function

Location:

- `contracts/delegates-handler-v1.clar`

The admin function `update-amounts()` has excessive powers as it can modify values for any delegate giving excessive permissions to any contract activated by the dao admins set.

Recommendation
Consider limiting the capabilities of this rescue function.

Status
Mitigated. The authentication/authorization of the function has been modified and now only the admin, passing as tx-sender, is authorized to call this function. However, the capabilities of this function remain.

ME-02 Failed Comparison of Delegates

Location:
- `clarity/contracts/version-2/strategy-v3.clar:195`

Classification:
- CWE-862: Missing Authorization

The function `strategy-v3::execute()` checks a list of up to 30 delegates received as input with a saved list with the aid of the function `compare-delegates()`. However, a bug in this function will return true in the presence of unequal lists when one is larger than the other one, and they agree in the common indices. As a result, a user could pass a delegates list which agrees with the saved one, plus adds delegates (repeating or not items in the list).

Recommendation
Make sure the validation function compares saved and input lists without errors. Implement tests.

Status
Fixed. The execute() function now additionally checks that the size of the lists matches in addition to checking that each entry matches.

Minor-Severity Issues

MI-01 Unrestricted Setter For A Limited Variable

Location:
- `clarity/contracts/version-2/data-direct-stacking-v1.clar`
- `clarity/contracts/version-2/stacking-dao-core-v2.clar`

Classification:
- CWE-1284: Improper Validation of Specified Quantity in Input

The function `set-direct-stacking-dependence(dependence)` is used to set a rate, which should be from u0 to u10000. However, the function allows setting any uint. An analogous issue lies in the stack-fee and unstack-fee in stacking-dao-core-v2.

Recommendation
Limit the input for this function. See also EN-03.

Status
Fixed. The constant `DENOMINATOR_BPS = u10000` was defined in this and the `strategy-v3-pools-v1.clar` contrat which uses `get-direct-staking-dependence()` to get the direct dependence. When setting the new dependence, the function asserts that the new value is smaller than this constant.

MI-02 Coarse Permission Checks Via check-is-protocol()

Location:
- `contracts/data-direct-stacking-v1.clar`

Classification:
- CWE-638: Not Using Complete Mediation

The function `check-is-protocol()` is used at several places in the contracts to check whether an address (for a deployed contract) has been whitelisted. This whitelist includes delegates, pools and many other contracts of different uses. Hence, this list is bound to grow and could include insecure or spurious contracts that may have an unforeseen impact.

Recommendation
We suggest documenting roles and permissions thoroughly and then having one “whitelist” per role, and giving specific permissions to this role. Say, the contracts in the delegates whitelist will include contracts that may act as delegates.

Status
Acknowledged. Developers mention that they decided not to change this as it adds more complexity and requires a malicious contract to be added to this whitelist. We mentioned that vulnerable and updatable contracts would also be liability.

Enhancements

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

ID	Title	Status
EN-01	Remove Development Comments	Implemented

ID	Title	Status
EN-02	Inefficient Implementations Resulting in Higher Gas Fees Paid	Implemented
EN-03	Prefer Using Named Constants Instead of uints	Implemented

EN-01 Remove Development Comments

Location:

- `contracts/version-2/strategy-v3.clar:231,239,`

It is a bad practice to have to-dos and other comments for developers that are not intended for documentation.

Recommendation

Remove to-dos and other comments for developers that are not for the arbitrary user.

Status

Implemented.

EN-02 Inefficient Implementations Resulting in Higher Gas Fees Paid

Location:

- `contracts/version-2/strategy-v3.clar:140,150.`
- `contracts/version-2/strategy-v3.clar:116,120,122.`
- `contracts/version-2/strategy-v3.clar:221.`

Function `prepare-delegates()` calls `get-preprare-pools-data(pool)` twice with the same input. The same happens with

- `get-pox-cycle()` which is called several times at this function or through functions called by this function. This pattern is repeated in other functions all of which results in unnecessary gas fees being paid.
- `prepare-pools()` calls map `map-pool-stacking-amount` that will call `get-pox-cycle()` repeatedly to obtain the same result.
- `return-unlocked-stx()` may call, via `return-unlocked-stx-helper()` and `delegates-handler-v1::handle-excess()` up to 30 times `dao::check-is-protocol(reserve)` for the same reserve.
- `get-unlock-burn-height()` is called once for every delegate received by `execute()` always returning the same value.

Recommendation
Revise code ensuring that there are no evidently unnecessary function calls..

Status
Implemented.

EN-03 Prefer Using Named Constants Instead of uints

Location:

- `clarity/contracts/version-2/strategy-v3-algo-v1.clar`

At several places in the code, values (uints) are used in place of constants. This could lead to problems when updating the code while using named constants will prevent this and improve readability. For example, the value u10000000000000 in strategy-v3-algo-v1.clar or the lists (list u1... u30), (list u0 u29).

Recommendation
Use named constants that are defined only once in the repository.

Status
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

The DAO contract through its admin functions is authorized to whitelist/blacklist delegates, pools and other parties, plus wields excessive power via delegates-handler-v1::update-amounts() function. These decisions should be documented to allow users to assess centralization risks.

Changelog

- 2024-05-12 – Initial report based on commit `a3b3f85169080276da11e984f0430a408394f5bf` and pull requests https://github.com/StackingDAO/StackingDAO/pull/220 and https://github.com/StackingDAO/StackingDAO/pull/213.

- 2024-05-20 – Reviewed changes for commit `f6b1f7f00bdf908a8fefe7eafd55c91dbe10e179`.

Disclaimer: This audit report is not a security warranty, investment advice, or an approval of the StakingDao's strategy v3 project since CoinFabrik has not reviewed its platform. Moreover, it does not provide a smart contract code faultlessness guarantee.