Stacks Audit

Proof of Transfer 4

March 2024

By CoinFabrik

v202311

Executive Summary

CoinFabrik was asked to audit the contracts for the Stacks project.

Stacks is a Bitcoin Layer-2, which enables smart contracts and decentralized applications to use Bitcoin as a secure base layer. Stacks extends the capabilities of Bitcoin without changing Bitcoin.

During this audit we found a minor issue. An enhancement was proposed.

The minor issue was acknowledged. The enhancement was not implemented.

Scope

The audited files are from the git repository located at https://github.com/stacks-network/stacks-core.git. The audit is based on the commit `5f01065c43cc88046880573b1b7ec5acf518257f`.

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

- `stackslib/src/chainstate/stacks/boot/pox-4.clar`: Core stacking contract. Mainly an update to PoX-3 with a focus on registering signers and removing rejection functionality.
- `stackslib/src/chainstate/stacks/boot/signers.clar`: Storage contract that holds the signer set and its associated data. It is updated every reward cycle.
- `stackslib/src/chainstate/stacks/boot/signers-voting.clar`: Contract which all signers use to vote on the next cycle aggregated public key.

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.

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 two 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 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
MI-01	Resolve TODO Comment	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

No issues found.

Minor Severity Issues

MI-01 Resolve TODO Comment

Location:

- `stackslib/src/chainstate/stacks/boot/pox-4.clar:896`

The previous location has the following comment

;; TODO: this must be authorized with a signature, or tx-sender allowance!

Recommendation

Implement the missing logic or remove the comment if it does not apply anymore.

Status

Acknowledged. The TODO comment will be removed in following iterations.

Enhancements

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

ID	Title	Status
EN-01	Use secp256k1-verify for Signatures Verification	Not implemented

EN-01 Use secp256k1-verify for Signatures Verification

Location:

- `stackslib/src/chainstate/stacks/boot/pox-4.clar:754-758`

The Proof of Transfer contract verifies the signature by recovering the public key used with `secp256k1-recover?` and then comparing it to the provided public key by parameter. However, the Clarity language already provides a native function for doing it in a single instruction, `secp256k1-verify`.

Recommendation

Simplify the function by using `secp256k1-verify`.

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

There is no point of centralization.

Upgrades

The Stacks node codebase needs to be updated in order to upgrade the contracts.

Privileged Roles

These are the privileged roles that we identified on each of the audited contracts.

Pool Delegate

Pool delegate / operator that stacks on behalf of other stackers (specifically 'pool stacker').

Signer

Known as a Signer, this is technically a node that has multiple signing responsibilities such as: signing for blocks, voting on an aggregated public key.

Node can Call Private Functions

In the signers contract (signers.clar), there are only private and read-only functions, so the contract state does not seem modifiable. However, only for this contract, private functions are called from the node via special boot code functionality where it can execute transactions directly on the clarity database.

Changelog

- 2024-03-08 – Initial report based on commit `5f01065c43cc88046880573b1b7ec5acf518257f`.
- 2025-01-17 – Final report based on development team feedback.