CoinFabrik

Zest Protocol Audit

Borrow

January 2024

By CoinFabrik

v202311

Executive Summary

CoinFabrik was asked to audit the contracts for the Zest Protocol project.

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

Three issues were resolved and two issues were mitigated. All the enhancements were implemented.

Scope

The audited files are from the git repository located at https://github.com/Zest-Protocol/zest-contracts/. The audit is based on the commit `dae42d8d6aa4710cab95bd44717a9dda40f2bd2e`. Fixes were checked on commit `8a206a8568acc5092e1a074e118fc4a8e9bbde4e`.

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

- `onchain/contracts/borrow/math/math.clar`: Math utility contract.
- `onchain/contracts/borrow/pool/fees-calculator.clar`: Calculates origination fee.
- `onchain/contracts/borrow/pool/oracle.clar`: Price oracle.
- `onchain/contracts/borrow/pool/pool-borrow.clar`: Interface for the system.
- `onchain/contracts/borrow/pool/liquidation-manager.clar`: Contract for liquidation functions.
- `onchain/contracts/borrow/vaults/pool-0-reserve.clar`: Main contract for managing pool state.
- `onchain/contracts/borrow/vaults/pool-reserve-data.clar`: Stores pool parameters.
- `onchain/contracts/borrow/token/zToken.clar`: Derivative from lended token.
- `onchain/contracts/borrow/traits/flash-loan-trait.clar`: Trait for flash loan executer.
- `onchain/contracts/borrow/traits/a-token-trait.clar`: Trait for token derivative.
- `onchain/contracts/borrow/traits/oracle-trait.clar`: Trait for oracle contract.

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. 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 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

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.

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
HI-01	No Maximum Value for Fees	High	Resolved
ME-01	Disabled Collateral Cannot be Liquidated	Medium	Mitigated

ID	Title	Severity	Status
ME-02	Authentication via tx-sender	Medium	Mitigated
MI-01	Panicking on Possible Error	Minor	Resolved
MI-02	As Contract Call to Unverified Principal	Minor	Resolved

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

HI-01 No Maximum Value for Fees

Location:

- `onchain/contracts/borrow/vaults/pool-reserve-data.clar: 81`

The origination fee can be set by any of the approved contracts from the system. This fee can be set calling `set-user-reserve-date()` with a new value. However, there is no maximum value for this fee. This can lead to an owner front-running users and raising the fees before they interact with the system. Also, borrowers might find a higher fee when repaying the loan, which could represent any value, even one higher than the borrowed amount.

Recommendation

Define a maximum value for the origination fee and enforce it.

Status

Resolved. The development team informed us that there is a global configurator and users will be warned of any changes to fees before they are executed so that front-running is not an issue for them before there is a confirmation of changes in the fees. Also, users will be able to set the maximum accepted fee on the website and it will be enforced by post-conditions.

Medium Severity Issues

ME-01 Disabled Collateral Cannot be Liquidated

Location:

- `onchain/contracts/borrow/pool/liquidation-manager.clar`
- `onchain/contracts/borrow/pool/pool-borrow.clar`

The `liquidation-call()` function can be called by any user in order to liquidate an undercollateralized position. However, this function reverts when the asset as collateral, either by the user or by the reserve.

For the user case, this does not represent an issue since the user cannot disable it while there is a position with that asset as collateral. While for the reserve, the asset can be disabled without any verification, but only the configurator role can do it through the `set-reserve()` function (located in `pool-borrow:432`).

Therefore, the configurator can disable an asset and get the system into an unsustainable economic state. In specific market conditions, this can lead to the protocol being unable to return the underlying asset to the suppliers.

Recommendation

The `set-reserve()` function should revert if the asset is in use. Otherwise, the decision of disabling an asset for a long period should be thoroughly investigated beforehand.

Status

Mitigated. Now a `frozen` state was added which is meant to be used only for emergencies. This status is set by the configurator. The issue persists, but only for assets in this status.

ME-02 Authentication via tx-sender

Location:

- `onchain/contracts/borrow/vaults/pool-0-reserve.clar`
- `onchain/contracts/borrow/token/zToken.clar`

The system utilizes `tx-sender` for its authentication processes. This method, while functional, presents latent vulnerabilities, particularly exposing actors within the system to threats known as phishing[1].

Actors could inadvertently activate a malicious contract. Once activated, the deceptive contract can access and initiate certain functions, presenting actions as if they were done by the original actor. This impersonation potential poses risks, depending on the specific function being accessed.

This issue has a larger impact when involving owner or configurator authentication because those users define system parameters and those calls are not protected by post-conditions as in asset transfers.

Recommendation

It is advisable to switch from using `tx-sender` to `contract-caller` for a more reliable and secure authentication method. It must be noted that when `tx-sender` is used as part of an `as-contract` invocation it does not lead to this issue, as it evaluates to the contract's principal.

Introducing a whitelist for trusted callers can add an extra layer of security, particularly if the system needs to interact with specific intermediary contracts. These intermediate

[1] https://www.coinfabrik.com/blog/tx-sender-in-clarity-smart-contracts/

contracts should properly check their `contract-caller` and/or pass it to the contract where the check needs to be made.

Status

Mitigated. The development team decided authentication via tx-sender must remain for minting, burning and transferring of assets to remain compatible with other assets in the Stacks ecosystem. Also, this risk would be properly documented for keeping users informed.

Minor Severity Issues

MI-01 Panicking on Possible Error

Location:

- `onchain/contracts/borrow/vaults/pool-0-reserve.clar: 225, 233, 238, 384, 1452`

Using `unwrap-panic` results in the transaction being finished because of a runtime error when the provided value is an error or a none. The runtime error does not allow the caller to handle that error and act in response. Also, this kind of error does not provide any information about the reason for the reverted transaction to the user.

While that form is a convenient method to unwrap values, it should not be used unless it is impossible to trigger the panic.

Recommendation

Replace `unwrap-panic` for `unwrap!` or `try!` when there is a flow which might trigger an error or none value.

Status

Resolved. Fixed according to the recommendation.

MI-02 As Contract Call to Unverified Principal

Location:

- `onchain/contracts/borrow/vaults/pool-0-reserve.clar: 1074`

Enclosing a contract call in an `as-contract` expression makes this internal call to be made on behalf of the caller contract (here `pool-0-reserve`). The `tx-sender` value is changed to this caller contract.

In a predefined set of scenarios, where the callees are limited to a whitelist, this does not involve an issue. However, here the principal (`asset`) is not verified. Then, any of the roles who can call this function (liquidators and lending pools, either as direct callers or as `tx-sender`) can use a malicious contract instead of an actual asset and use `pool-0-reserve` authorization for modifying protocol state in unexpected terms.

Currently, this issue is minor because the `pool-0-reserve` contract only has authorization for functions which authenticate through `contract-caller`. Then, no malicious contract will work. However, this should be considered for newer contracts, which might be vulnerable to this issue.

Recommendation
Validate the actual parameter against a whitelist.

Status
Resolved. Function removed.

Enhancements

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

ID	Title	Status
EN-01	Remove Dead Code	Implemented
EN-02	Place Definitions before Usage	Implemented
EN-03	Resolve TODO Comments	Implemented

EN-01 Remove Dead Code

Location:

- `onchain/contracts/borrow/pool/pool-borrow.clar: 12, 81, 111`
- `onchain/contracts/borrow/vaults/pool-0-reserve.clar: 19, 349, 1589, 1612, 1635`
- `onchain/contracts/borrow/pool/fees-calculator.clar: 5, 9, 17, 21 (user parameter)`
- `onchain/contracts/borrow/math/math.clar: 34, 40`

The listed code lines have functions, parameters, values or commented code which are not used. This reduces code readability.

Status

Implemented.

EN-02 Place Definitions before Usage

Location:

- `onchain/contracts/borrow/pool/liquidation-manager.clar`
- `onchain/contracts/borrow/token/zToken.clar`
- `onchain/contracts/borrow/vaults/pool-0-reserve.clar`
- `onchain/contracts/borrow/vaults/pool-reserve-data.clar`
- `onchain/contracts/borrow/math/math.clar`

Listed files contain definitions, mostly error definitions, which are placed at the bottom of the file, after their usage instances. It is preferable to define them before for better readability.

Status

Implemented.

EN-03 Resolve TODO Comments

Location:

- `onchain/contracts/borrow/token/zToken.clar`

In the listed files, there are comments with TODOs which should be either done or removed.

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

There are two important roles which are the contract owner and the configurator. Those two can set different system parameters which alter the functionality of the application.

Upgrades

Contracts are tightly coupled. There is no mechanism for upgrading them, with the exception of the treasury and the reserve vaults.

Changelog

- 2024-02-01 – Initial report based on commit `dae42d8d6aa4710cab95bd44717a9dda40f2bd2e`.
- 2024-02-15 – Fixes checked on commit `8a206a8568acc5092e1a074e118fc4a8e9bbde4e`.
- 2024-02-19 – ME-02 severity modified and status updated based on feedback from the development team.