Zest Audit

August 2022

By CoinFabrik

Introduction

CoinFabrik was asked to audit the contracts for the Zest Protocol project. First we will provide a summary of our discoveries, and then we will show the details of our findings.

Scope

The audited files are from the git repository located at https://github.com/y0s0n/zestAudit. The audit is based on the commit `c94cd6fdc98effcea10329fff99da2f0fb471479`.

The audited files are:

- `./contracts/globals.clar`
- `./contracts/protocol-treasury.clar`
- `./contracts/rewards-calc.clar`
- `./contracts/swap-router.clar`
- `./contracts/loan/pool-v1-0.clar`
- `./contracts/loan/cover-pool-v1-0.clar`
- `./contracts/loan/payment-fixed.clar`
- `./contracts/pool/cover-pool-v1-0.clar`
- `./contracts/token/xZest-token.clar`
- `./contracts/token/cp-token.clar`

The scope of the audit is limited to those files. 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.

Analyses

Without being limited to them, the audit process included the following analyses:

- Arithmetic errors
- Race conditions
- Reentrancy attacks
- 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

Summary of Findings

We found 2 medium-severity issues and 3 minor-severity issues. Also, several enhancements were proposed.

Security Issues

ID	Title	Severity	Status
ME-01	Double Borrowing May Be Allowed	Medium	Unresolved
ME-02	Using tx-sender For Authentication Is Discouraged	Medium	Unresolved
MI-01	Staker Funding A Pool Twice Counted Once	Minor	Unresolved
MI-02	Independent Loan Creation, Funding and Unwinding Amounts	Minor	Unresolved
MI-03	Floor Rounding When Computing Collateral Amount	Minor	Unresolved

Security Issues Found

Severity Classification

Security risks are classified as follows:

- **Critical:** These are issues that we manage to exploit. They compromise the system seriously. They must be fixed **immediately**.
- **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 can 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 can have four distinct statuses:

- **Unresolved**: The issue has not been resolved.
- **Acknowledged**: The issue remains in the code, but is a result of an intentional decision.
- **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 critical issues found.

Medium Severity Issues

ME-01 Double Borrowing May Be Allowed

After a borrower has created a loan, the pool delegate can fund the loan with an arbitrary amount. The borrower can then call `pool-v1-0::drawdown()` as many times as funding allows, since his calling `drawdown()` does not deactivate the loan.

Recommendation

Before allowing a borrower to draw down, check that the loan has not been drawn down before.

Status

Unresolved.

ME-02 Using `tx-sender` For Authentication Is Discouraged

Methods throughout the code base use the keyword `tx-sender` for authentication, which is discouraged as it returns the original sender of the current transaction. A valid user could inadvertently fall victim to a malicious contract, e.g., via a phishing

attack, and involuntarily execute one of these methods. In particular in functions like this

```
(define-public (set-contract-owner (owner principal))
 (begin
   (asserts! (is-eq tx-sender (var-get contract-owner))
ERR_UNAUTHORIZED)
   (ok (var-set contract-owner owner))
 )
)
```

Also `cover-pool-v1-0::is-staker()`, `cover-pool-v1-0::withdraw()`, `xZest-tokens::transfer()`, `xZest-tokens::withdraw-rewards()`

Recommendation

Prefer using the keyword `contract-caller`.

Status

Unresolved.

Minor Severity Issues

MI-01 Staker Funding A Pool Twice Counted Once

The function `cover-pool::send-funds()` could be called twice from the same staker, overwriting `sent-funds[owner, token-id] = {start: block-height, withdraw-signaled: 0}`.

Recommendation

Make sure that either double-calling this function is disabled or that funds additions are recorded correctly. Also, provide documentation for stakers.

Status

Unresolved.

MI-02 Independent Loan Creation, Funding and Unwinding Amounts

A borrower establishes the amount for his loan when calling `loan-1-0::create-loan()`. Next, the pool delegate will call `::fund-loan()` authorizing the loan and defining an independent amount that is transferred from the liquidity vault to the funding vault of this loan. Now the borrower may access the loan through the supplier interface and get the amount he requested on loan creation, or the pool delegate can undo this transaction, calling `pool-v1-0::unwind()`, and specifying a third amount, which must be bigger than

or equal to the first (loan) amount. Once `::unwind()` is called once, the loan is set to `EXPIRED` and neither the borrower can claim a loan nor the pool delegate can call unwind again.

Hence, it may be the case that the borrower requests a loan of 100, it gets funded by 150 from the pool delegate and then the pool delegate unwinds for 100, missing 50.

Recommendation
Design a data structure that allows recording the loan state thoroughly throughout the contracts. Also re-use the amount from records.

Status
Unresolved.

MI-03 Floor Rounding When Computing Collateral Amount

When a borrower calls the `loan-v1-0::drawdown()` to get the BTC from his loan, the collateral amount is computed on the spot as

```
coll-amount = coll-ratio * loan-amount / 10000
```

Of course, Clarity is going to floor-round the division, possibly cropping as much as 9999 from `coll-amount`.

Recommendation
Use `coll-amount = ((coll-ratio * loan-amount) + (10000 - 1) ) / 10000` if you want to use ceiling rounding.

Status
Unresolved.

Enhancements

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

ID	Title	Status
EN-01	Ownership Checks Not Using Special Function	Not implemented
EN-02	Dead Code	Not implemented
EN-03	Prefer Constants Instead of uint For Error Codes	Not implemented

EN-04	Gas Saving Opportunities	Not implemented
EN-05	Misleading Function Naming	Not implemented
EN-06	Consider Upper-Bounding maturity-length	Not implemented
EN-07	Per Pool Borrower Authorization	Not implemented

EN-01 Ownership Checks Not Using Special Function

Location:

- `contracts/globals.clar:339, 346.`

Functions like `globals::onboard-user()` and `globals::offboard-user()` include the code

```
(asserts! (is-eq tx-sender (var-get contract-owner)) ERR_UNAUTHORIZED)
```

`instead of calling the ::is-contract-owner()` function.

Recommendation

Always define and use specific functions for authentication.

Status

Not implemented.

EN-02 Dead Code

Some smart contracts include methods or variable declarations that are unused. Consider removing them in order to save gas and have a cleaner code base.

1. Remove unused variable from `cover-pool-v1-0.clar`
   ```
   (define-data-var enabled bool true)
   ```
2. Remove dead code, e.g., `liquidity-vault-v1-0::fund-loan()`
3. Unused data map in `tokens/cp-tokens.clar L340`
   ```
   (define-map rewards { token-id: uint, cycle: uint} uint)
   ```
4. The constant `BITCOIN_PRECISION` is defined in 8 contacts, but never used.
5. Unused var in `tokens/lp-token.clar::withdraw-rewards(), L156`
   `(recipient-contract contract-caller)`.
6. In pool-v1-0::create-pool() the variable globals is defined but not used in L94
   ```
   ((globals (contract-call? .globals get-globals)))
   ```

Status
Not implemented.

EN-03 Prefer Constants Instead of `uint` For Error Codes

Use constants for errors in `cover-pool::withdraw()` not `err u999`, and maybe use different errors.

Moreover, at several places in the code, the same constants are used but they are not declared as such in the contract's preamble. Prefer declaring them, e.g, so that they can be modified in new deployments without causing trouble. For example, `rewards-calc::L31`, or the polynomial definition in that contract and `pool-v1-0::set-delegate-fee() L214`.

Similarly, in `swap-router.clar` there are addresses and constants used throughout the contract which should be defined explicitly.

Status
Not implemented.

EN-04 Gas Saving Opportunities

1. Save gas in `cover-pool::withdraw()` reversing the order of these commands and reusing the variable.

```
(withdrawal-time-delta (- block-height (get withdrawal-signaled
sent-funds-data)))

(withrawal-signaled-time (get withdrawal-signaled sent-funds-data))
```

2. In `pool-v1-0::send-funds()` the second check is superfluous

```
(asserts! (is-eq (get status pool) READY) ERR_POOL_CLOSED)

(asserts! (not (is-eq (get status pool) DEFAULT)) ERR_POOL_DEFAULT)
```

3. Authorization/access control code in `pool-v1-0::accept-rollover()` is repeated

```
(try! (tx-sender-is (get pool-delegate pool)))

(asserts! (is-eq tx-sender (get pool-delegate pool)) ERR_UNAUTHORIZED)
```

4. More generally, consider factoring out access control checks into a single function, e.g., that checks if the caller is the pool owner and the contracts are paused, etc.
5. The check `new-lc > 0` in the function `pool-v1-0::lc-check()` is superfluous

```
(define-read-only (lc-check (new-lc uint) (previous-lc uint))
      (and (> new-lc previous-lc) (> new-lc u0))
)
```

Given that previous-lc must be at least 0.

Status
Not implemented.

EN-05 Misleading Function Naming

Functions `cover-pool-v1::is-enabled()` and `pool-v1-0::is-paused()` have the same functionality, but a different name. Moreover, is-paused() returns with an error if the contact is paused.

Status
Not implemented.

EN-06 Consider Upper-Bounding maturity-length

The length in which a borrower pays back his loan is defined as maturity-length, selected by the borrower and accepted by the pool delegate (who must explicitly call `pool-v1-0::fund-loan()`). Nonetheless, allowing the pool delegate to limit duration may help communicate his preferences.

Status
Not implemented.

EN-07 Per Pool Borrower Authorization

Borrowers are onboarded globally by `globals` contact owner. But risks may differ from pool to pool. Any onboarded borrower can take loans from any pool/

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 for other stakeholders of the project, including users of the audited contracts, owners or project investors.

Upgrades

Upgrades seem to be considered partially. Some smart contracts, for example, allow setting a new pool contract. However, for example, there is not one single method to replace the pool contract in each of the contrats it is used, and in cases like rewards-calc the pool-v1-0 address cannot be modified.

Tests, Documentation and Comments

Test coverage is partial. Most of the functionality for pool creation, loan funding, borrowing and staking is not covered by tests.

Also consider adding documentation for the main functionalities. In particular, it is important to define authorization and requirements for pool creation, loan funding, unwinding, and other functions.

It often happens that relevant functions are prepended with a comment including documentation. But this is not done consistently, and in some cases (e.g. `pool-v1-0::drawdown()`) there are function parameters that are not described and some parameters which are described are not function parameters.

Mocks

The contract `swap-router` is a mock, and although part of the scope, it is basically empty and lacks the code that is liable to security issues, e.g., through the introduction of oracles or complex pricing schemes. Consider using only trusted swaps when replacing this contract.

Privileged Roles

Owner

The contracts: globals, protocol-treasury, rewards-calc, pool-v1-0

The owner is set by the following line:

```
(define-data-var contract-owner principal tx-sender)
```

but can be replaced. This owner can perform critical operations and its private keys should be guarded heavily.

Approved Contracts

Some of the methods in certain contracts can only be called by a set of specific contracts. For example, `loan-v1-0::create-pool()` can only be called by the pool contract, which is set to `pool-v1-0`. This, together with the code of the calling contracts, defines very specifically how these functions are called and which parameters are used.

Pool Delegate

The owner of the pool contract can create pools, through create-pool(), and define a principal as pool delegate. This principal is the only entity authorized to modify pool parameters (e.g., liquidity cap, cycle length), modify the pool's state (set open, finalize), and fund, liquidate or unwind loans.

Changelog

- 2022-08-19 – Initial report based on commit `c94cd6fdc98effcea10329fff99da2f0fb471479`.

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