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 audit was divided into three iterations, working with three different versions of the source code. In the first two iterations, we looked for security vulnerabilities and in the third one we checked that the resolution of the issues was made.

After this audit was finished, the development team uploaded the audited code to the https://github.com/Trust-Machines/zest-contracts.git git repository.

It must be noted that all the hashes mentioned below were calculated by running the `sha256sum` utility.

First Iteration

For the first iteration the following files, with the following sha256 hashes, were analyzed:

- `contracts/loan/coll-vault.clar:`
  `53a5228fc63ce01d52910b02877ec079f2a7ff11f0f829670fa190bb6a3f8220`
- `contracts/loan/funding-vault.clar:`
  `070ca664873443980d9b144e0a587b1a5f84e4592b654a68a313f28fe5fdd1e3`
- `contracts/loan/loan-v1-0.clar:`
  `08424d282eb2bdd9087b7af23d7a4d483b9070c525bbe9b073afaa18b11846d2`
- `contracts/loan/payment-fixed.clar:`
  `0a07afe5df23d95d0cf8d38ab01cfdff29c88d5f9ed0a21fe94a571ebe98d336`
- `contracts/pool/liquidity-vault-v1-0.clar:`
  `d32b966d20d6260d485318660ff79db1f2726b1e4beb6894ae7c5362a46d5178`
- `contracts/pool/pool-v1-0.clar:`
  `74c01ff4ee3d9a4ed241f0f52619722b5bca615b8d14e0886dfdaed12fb60e0f`
- `contracts/pool/staking-pool-v1-0.clar:`
  `4646b4622a5197df477be26f01c34d9f0b936f5ad325c40af2480487431ef445`
- `contracts/rewards-multiplier.clar:`
  `d502fd79c43060f140fb3ac652bf600ad80c1b4795ba85709da2eb4265779ebe`
- `contracts/token/lp-token.clar:`
  `db34f5727baa5176fd4d4c3073d88a81d9e5e7645d39b7a7b06f1d27d744ed09`

- `contracts/token/sp-token.clar:`
  `14f2101d5a1b9d5c1405f5d3f50e9a311d54d5ea57ca8b3416f4b17ceccbc2c0`
- `contracts/token/xbtc.clar:`
  `e992b1dc9034c45e3a9007ef92a462d83cb1608a9393573c88a7b6f5c88120ec`
- `contracts/token/zest-reward-dist.clar:`
  `a235defa364ff1bfe070c9fb1e6f043d3560860935e24ce315ddce4c66a36f3c`

The scope of the iteration 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 iteration.

This iteration corresponds to the commit `b8158372ec05069ceccbb939f48f3ac2bf31e8de` in the git repository.

Second Iteration

For the second iteration the following files, with the following sha256 hashes, were analyzed:

- `contracts/globals.clar:`
  `989ab9d5c955b4208abe9d93f5e7b179f61a672e932db3b22c58a370f0b3c782`
- `contracts/loan/payment-fixed.clar:`
  `e40a9fb72e177ebcdb87c7bc00e5ae0885c0631aee73a07aa7ec4b55a576cae8`
- `contracts/pool/cover-pool-v1-0.clar:`
  `aa385c13f848d1fd2a37ca1779b81da582244846a80863a05adc8371e83c83fe`
- `contracts/pool/pool-v1-0.clar:`
  `4dff56c137468a31b4140f654d357609b2f511497debb6155a88487b686c8e9f`
- `contracts/protocol-treasury.clar:`
  `885adab5fec9f130f33c15a0e27872f691a8106b62a6a58861aa990579f1ec25`
- `contracts/rewards-calc.clar:`
  `f6df8dbb069f67da2354db6080642dcf9e06cc30cef0e07079bd20302ebcb86f`
- `contracts/swap-router.clar:`
  `5788ce1dc72babbd99945c4a83de8de87a0b9ca1d2d68fece086c7c890dcc31f`
- `contracts/token/cp-token.clar:`
  `a5c660c8a8c9d4b20bcc6646377b8dde001bd15264667cf085c495221a93c792`
- `contracts/token/xZest-token.clar:`
  `b553ae6db1ca31fae5f35f8e83f21670caac755a42bffd130d329b3f6d5e77f0`

The scope of the iteration 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 iteration.

This iteration corresponds to the commit `4e3829dcaffcbe4214c3a96fae1c5aa975d187cd` in the git repository.

Fixes

The third iteration was used to check that the issues and enhancements found in the first two iterations were properly resolved.

The following files, with the following sha256 hashes, were analyzed:

- `contracts/globals.clar:`
  `d6954abfda2f12083e33177a5f91e44d11bb8cea89332a9cc239aaea6c8a063f`
- `contracts/loan/coll-vault.clar:`
  `80ae38aca48c4e6652c2a33a9a9ca2af238608f3a1c863f0dc048eb9dd3e9d0e`
- `contracts/loan/funding-vault.clar:`
  `8bca190ad6cd8ec10bbb32501983342de4c5043361614f62d619edf0b4fee777`
- `contracts/loan/loan-v1-0.clar:`
  `b31e683dadb81e805919f5ef15eddceed4e07c1affdc7326c4fda2879f5afbc9`
- `contracts/loan/payment-fixed.clar:`
  `e40a9fb72e177ebcdb87c7bc00e5ae0885c0631aee73a07aa7ec4b55a576cae8`
- `contracts/pool/cover-pool-v1-0.clar:`
  `2e8d4a1b60070d74786815f41d5ea92cbd1a026a77f2510d4d2a995b99be00c9`
- `contracts/pool/liquidity-vault-v1-0.clar:`
  `c27a809df3f57e9009ca9b3113cfac74f2ad8a36f4484f338be75f23a3728311`
- `contracts/pool/pool-v1-0.clar:`
  `fa6ca9c52dc76a8c6d88b42c23e61e33bc5c43bf6b5eb6d36e5049c1a8eab92f`
- `contracts/protocol-treasury.clar:`
  `885adab5fec9f130f33c15a0e27872f691a8106b62a6a58861aa990579f1ec25`
- `contracts/rewards-calc.clar:`
  `942faf83adb219da12ef2b0fbb3bc4d9838280a2f9fbdcf642cf946468bb65ec`
- `contracts/swap-router.clar:`
  `0e0fbc61ce8479941d38fe0203608e009dc1cf0131eb6d56eae54b2b99928786`
- `contracts/token/cp-token.clar:`
  `9b4cfd9b446d7866abc422be73881583af1d97340d2722c0ae8b026af89f95e7`
- `contracts/token/lp-token.clar:`
  `b21981345c981c7952209df78792df15b0468125631a88e3761ad36e6ec3c4f4`
- `contracts/token/xbtc.clar:`
  `24d6e20189580a26832b44f53511af8e5d946a386db8f86744c4a61a6ffbd346`
- `contracts/token/xZest-token.clar:`
  `41f0f4650b99774af079d7b44d551addcf5eb3e1d7b10965df2f6a7dba974d04`

- `contracts/token/zest-reward-dist.clar`: `4f3beef7857273b122c919c2a28a134b8ff9a573940844c308221c3fe788fa94`

It also must be noted that the following files, which were analyzed in the first iteration, were removed when we checked for the fixes:

- `contracts/pool/staking-pool-v1-0.clar`
- `contracts/rewards-multiplier.clar`
- `contracts/token/sp-token.clar`

This iteration corresponds to the commit `244e3b2c0aa156afc04a844f67ad2e781b651075` in the git repository.

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

In the first iteration found one critical issue, three medium issues and three minor issues. Also, several enhancements were proposed.

In the second iteration, we found 2 medium-severity issues and 3 minor-severity issues. Also, several enhancements were proposed.

All issues of the first and second iterations were resolved.

Security Issues

ID	Title	Severity	Iteration	Status
CR-01	Wrong xBTC Address	Critical	First	Resolved
ME-01	Unrestricted Mint and Burn By Owner	Medium	First	Resolved
ME-02	Owner May Input Negative Rewards	Medium	First	Resolved
ME-03	Unvalidated Transfers in deposit() and deposit-rewards()	Medium	First	Resolved
ME-04	Double Borrowing May Be Allowed	Medium	Second	Resolved
ME-05	Using tx-sender For Authentication Is Discouraged	Medium	Second	Resolved
MI-01	Removing Disapproved Contracts	Minor	First	Resolved
MI-02	Frontrunning Lockup Period	Minor	First	Resolved
MI-03	Deposit Date Delayed When Deposit Called Twice	Minor	First	Resolved
MI-04	Staker Funding A Pool Twice Counted Once	Minor	Second	Resolved
MI-05	Independent Loan Creation, Funding and Unwinding Amounts	Minor	Second	Resolved
MI-06	Floor Rounding When Computing Collateral Amount	Minor	Second	Resolved

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

CR-01 Wrong xBTC Address

Iteration: First

Throughout the contracts there are several references to a XBTC contract, and in particular, the transfer function from the bridge contract makes a call to a contact `xbtc.clar` contract:

```
contract-call? .xbtc transfer amount sender recipient none
```

which is not the xBTC (external) contract, but a contract with the same name defining a homonymous token. Deploying the contracts as in this commit could lead to confusion and having unprevented users swap their BTC for this token with a few hard-coded addresses.

Recommendation

Remove the xbtc.clar contract and use the xBTC transactions instead.

Status
Resolved. The calls are now transferring XBTC tokens.

Medium Severity Issues

ME-01 Unrestricted Mint and Burn By Owner

Iteration: First

In the contracts `loan-token`, `lp-token`, `sp-token`, `zest-reward-dist` it is either the owner or an approved contract who can mint new tokens to anyone or burn any user's tokens. It is preferable that only a predefined set of contracts does this. Users should not put their trust in any party, who may be compromised or act maliciously.

Recommendation:
Use specific functions that mint or burn tokens on conditions that are public and known to users.

Status:
Resolved. Mint and burn calls are now wrapped with specific functionalities that are part of the protocol and can only be called by approved callers. These callers are a static list of smart contracts that are defined within the contract.

ME-02 Owner May Input Negative Rewards

Iteration: First

In the `loan-token` contract, the function

```
(define-public (deposit-rewards (amount int))
 (begin
   (asserts! (is-contract-owner contract-caller) ERR_UNAUTHORIZED)
   (ok (update-funds-balance amount))
 )
)
```

can be called with any integer (positive or negative) and calls the private function

```
(define-private (update-funds-balance (delta int))
 (begin
   ;; update distribution funds in holdings
   (var-set funds (to-uint (+ (to-int (var-get funds)) delta)))
   (asserts! (and (> delta 0 ) (> (ft-get-supply loan) u0)) delta)
   ;; only distribute when funds are earned
   (var-set points-per-share (+ (var-get points-per-share) (/ (*
(to-uint delta) POINTS_MULTIPLIER) (ft-get-supply loan))))
   delta
```

```
  )
)
```

It seems that the first function should only be called using positive integers. Something similar happens in `lp-token::deposit-rewards()`, `sp-token::deposit-rewards()` and `zest-rewards-dist::deposit-rewards()`.

Recommendation
Either require the value to be a positive integer, or use input of type uint.

Status
Resolved. The logic has been changed, these functions are no longer part of the contracts.

ME-03 Double Borrowing May Be Allowed

Iteration: Second

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
Resolved. The `drawdown()` function was changed to require the loan status to be INIT and it changes the status to ACTIVE, thus logically preventing the function from being successfully called more than once.

ME-04 Using tx-sender For Authentication Is Discouraged

Iteration: Second

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

Resolved. Changes throughout the code reflect the change. The keyword `tx-sender` is still maintained in a few specific cases where it is needed.

Minor Severity Issues

MI-01 Removing Disapproved Contracts

Iteration: First

The contracts `rewards-multiplier, payment-fixed liquidity-vault-v-0`, `staking-pool-v1-0`, `lp-token`, `sp-token` and `zest-reward-dist` implement the same function `remove-contract()` to remove contracts from the list of approved contracts. This function simply checks if the caller is authorized and then sets the principal in the input to disapproved. The to-be-disapproved contract could already be disapproved, so the owner calling the function may think a change was made after calling the function when actually no change was made.

The same happens with `loan-v1-0::remove-borrower()` which may be called to remove a nonexistent borrower.

Status

Resolved. These functions have been removed from the contracts.

MI-02 Frontrunning Lockup Period

Iteration: First

There is a race condition with a liquidity provider calling `pool-v1-0::deposit()` and the delegate calling `pool-v1-0::set-lockup-period()` right before. This could have a liquidity provider's assets locked up for a longer time than he expected.

Recommendation

Allow the user calling deposit to set the lockup he wants to get or a threshold on the lockup, and then compare with the value set by the delegate.

Status
Resolved. Logic has been changed and this is no longer possible.

MI-03 Deposit Date Delayed When Deposit Called Twice

In `pool-v1-0::deposit()` the same liquidity provider (`tx-sender`) could call the function more than once, but since the deposit-date is a map keyed by `tx-sender`, the last call is going to overwrite the preceding ones.

Recommendation
Make sure the deposit date is not overwritten by modifying the map, e.g., the deposit date could be mapped to a deposit id, and withdraw() can loop over the different deposits or be directed to withdraw from one specific deposit.

Status
Resolved. Function was removed.

MI-04 Staker Funding A Pool Twice Counted Once

Iteration: Second

Location:

- `contracts/pool/cover-pool-v1-0.clar:115`

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
Resolved. The sent-funds map now is updated using the new get-new-factor() which computes values taking into account the previous value of the map.

MI-05 Independent Loan Creation, Funding and Unwinding Amounts

Iteration: Second

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

Resolved. The loan amount is set once and then retrieved from storage.

MI-06 Floor Rounding When Computing Collateral Amount

Iteration: Second

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

Resolved. The logic was changed so that the function reverts when `coll-amount` is 0.

Enhancements

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

Table

ID	Title	Iteration	Status
EN-01	Best Practices in Constants, Variables and Maps Definitions	First	Implemented
EN-02	Comments Not Intended For Users	First	Partially implemented
EN-03	Ownership Checks Not Using Special Function	Second	Implemented
EN-04	Dead Code	Second	Implemented
EN-05	Prefer Constants Instead of uint For Error Codes	Second	Implemented
EN-06	Gas Saving Opportunities	Second	Implemented
EN-07	Misleading Function Naming	Second	Implemented
EN-08	Consider Upper-Bounding maturity-length	Second	Implemented
EN-09	Per Pool Borrower Authorization	Second	Not implemented

Details

EN-01 Best Practices in Constants, Variables and Maps Definitions

Iteration: First

Throughout the contracts, constants, variables and maps are defined all over each contract instead of preferring the first lines of the contract and adding some documentation through comments.

Recommendation

Use the preamble of the contract for defining variables, constants and maps.

Status

Implemented.

EN-02 Comments Not Intended For Users

Iteration: First

In the contract `lp-token` L118 there is a commented function that should be removed

```
;; (define-public (call-this)
;;   (ok {first: contract-caller, result: (is-contract-owner contract-caller),
second: (get-contract-owner)})
;; )
```

The same happens at L258 of the same contact

```
;; (define-private (update-funds-losses (delta int))
;;   (let (
;;     (prev-losses (var-get fund-losses))
;;   )
;;     (var-set fund-losses (to-uint (+ (to-int prev-losses) delta)))
;;     delta
;;   )
;; )
```

Also in pool-v1-0 L 99 and L197. Malicious users may use to-do lists, commented code or other information provided within comments to their aid.

Recommendation

Remove all commented code and more generally comments not directed to documenting functions or aiding the reader.

Status

Partially implemented. Many of these occurrences have been removed, although some still persist.

EN-03 Ownership Checks Not Using Special Function

Iteration: Second

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

Implemented.

EN-04 Dead Code

Iteration: Second

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:340`
   ```
   (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()`, (line 156) `(recipient-contract contract-caller)`.
6. In `pool-v1-0::create-pool()` the variable globals is defined but not used in line 94 `((globals (contract-call? .globals get-globals)))`

Status

Implemented.

EN-05 Prefer Constants Instead of `uint` For Error Codes

Iteration: Second

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

Implemented.

EN-06 Gas Saving Opportunities

Iteration: Second

1. Save gas in `cover-pool-v1-0::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
Implemented.

EN-07 Misleading Function Naming

Iteration: Second

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
Implemented. Some functions were renamed to get consistency.

EN-08 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

Implemented. The parameter `max-maturity-length` was added to `pool-v1-0::create-pool()` and `pool-v1-0::create-loan()` in order to limit the maturity length of a loan.

EN-09 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.

Status

Not Implemented. This is a design decision: “The desired functionality is that borrowers can borrow from multiple pools once onboarded.”

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 – Report with second iteration of the audit.
- 2022-09-05 – Check fixes for the second iteration of the audit.
- 2022-09-29 – Merge first iteration of the audit in the same report.
- 2022-10-04 – Add reference to new git repository with the audited code in the scope section.

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.