CoinFabrik

AlexGo Audit

Launchpad, Vault, and Reserve Pool

January 2022

By CoinFabrik

Introduction

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

Scope

The contracts audited are from the alex-v1 git repository. The audit is based on the commit `44c44846bfbcce6096be04bd1380728c98f09ec8`. The fixes were added to the commit `31a5d660c83d41c10cd1b34498f02bc3a407721e`.

The audited contracts are:

- `clarity/contracts/alex-vault.clar`: Contract that stores system tokens and allows flash loans.
- `clarity/contracts/pool/alex-launchpad.clar`: IDO token launchpad.
- `clarity/contracts/pool/alex-reserve-pool.clar`: Contract for token staking.

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 a critical issue, two medium issues and a minor issue. Also, two enhancements were proposed.

The critical severity issue and the two medium issues were acknowledged. Two minor severity issues were fixed. An enhancement was implemented.

Security Issues

ID	Title	Severity	Status
CR-01	Unfair Lotteries through Weak Randomness	Critical	Acknowledged
ME-01	Insecure Authentication through tx-sender	Medium	Acknowledged
ME-02	Independent Winning Probability in Lottery	Medium	Acknowledged
MI-01	Arithmetic Underflow Calculating Staking Reward	Minor	Resolved
MI-02	Ended Pool can be Created	Minor	Resolved

Privileged Roles

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

Alex-vault.clar

Owner

In the beginning, the owner is the address of the deployer. Then, the owner can set an address as a new owner. Also, this role can set a new flash loan fee rate and add new approved contracts, flash loan users and flash loan tokens. Finally, the owner can transfer fungible and semi-fungible tokens stored in the vault.

Approved Contracts

The approved contracts are addresses which can execute the transfer functions to move fungible and semi-fungible tokens from the vault contract. This address set is initialized including `alex-reserve-pool`, `collateral-rebalancing-pool`, `fixed-weight-pool`, `liquidity-bootstrapping-pool`, `yield-token-pool`, and `yield-collateral-rebalancing-pool`.

Approved Flash Loan Users

Flash loan users are addresses allowed to be used when `flash-loan()` is called. This contract should implement the specific trait in order for the vault contract to call the `execute()` function.

Alex-launchpad.clar

Owner

In the beginning, the owner is the address of the deployer. Then, the owner can set an address as a new owner. Also, this role can create new token-ticket pools.

Liquidity Provider

This is an address set by the owner for each pool created. This address is the only one allowed to provide tokens to the pool. Also, this address receives the amount of stacks paid to validate the winner ticket.

Alex-reserve-pool.clar

Owner

In the beginning, the owner is the address of the deployer. Then, the owner can set an address as a new owner. Also, this role can add new approved contracts and approved tokens, set a new activation delay, activation threshold, a new value for the halving cycle and coinbase amount of a token and a new reward cycle length. Finally, the owner can increase and decrease the balance of a token.

Approved Contracts

The approved contracts are addresses which can increase and decrease the balance of a token. Initially, the contracts included in this set are: `collateral-rebalancing-pool`, `fixed-weight-pool`, `yield-token-pool`, `yield-collateral-rebalancing-pool` and the reserve pool itself.

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.
- **Mitigated**: Implemented actions to minimize the impact or likelihood of the risk

Critical Severity Issues

CR-01 Unfair Lotteries through Weak Randomness

Location:
- `clarity/contracts/pool/alex-launchpad.clar`

Users can validate their tickets during the registration process, delimited by the variables `registration-start` and `registration-end`. A counter is increased for each ticket validated, and each user has a range of positions related to when they joined the lottery and the amount of tickets validated. When the registration ends and the minimum amount of participants is reached, users can call `claim()` one time for each ticket validated. This function determines if the ticket is a winner based on a pseudo-random number modulo the counter of tickets. If the resulting value is in the range of positions of tickets validated by the user, then it is a winner ticket.

However, the first pseudo-random number is generated based on the vrf-seed of `registration-start` block in the first call to `claim()`. For the subsequent calls, the function calculates a new pseudo-random based on the latest random. Therefore, since the contract is public and the process transparent, anyone can calculate the sequence of values to be generated. Users can speculate with which position is the most convenient to register themselves and when to claim based on the following number in the sequence.

Recommendation

The speculation in the registration can be solved using the VRF seed of the block next to the end of the registration (`registration-end + 1`) as randomness source.

Finally, the speculation in the claiming order can be solved computing the pseudo-random number with the mentioned VRF seed and a value unique for each ticket (e.g., the ticket's position). Then, using two constant values, the claiming order will not have an impact because the random number is already determined.

Status

Acknowledged. `claim()` now uses `registration-end` instead of `registration-start` for the VRF seed. The speculation was reduced, but it still uses a seed that can be known if `register()` is called at `registration-end`. Using the block next to `registration-end` can solve the problem.

The new changes also made the entire random sequence unpredictable. However, the next random still can be predicted and the user might check if this random is beneficial and wait for the next random if not. In order to avoid the speculation in the claiming order, the number should be generated with either an input unknown for the user or constant values. The first solution makes the next random unpredictable for the user, while the second one makes it constant

Medium Severity Issues

ME-01 Insecure Authentication through tx-sender

Location:

- `clarity/contracts/alex-vault.clar`,
- `clarity/contracts/pool/alex-launchpad.clar`,
- `clarity/contracts/pool/alex-reserve-pool.clar`

Global variable `tx-sender` returns the original sender of the current transaction, or if `as-contract` was called to modify the sending context, it returns that contract principal. Using this variable for authentication is not secure. Actors in the system could be targets of phishing. This is analogous to what happens to `tx.origin` and `msg.sender` in Solidity. There, the convention is to use `msg.sender`, which works like the `contract-caller`.

For instance, the vault's owner can be tricked into calling a malicious contract which executes `vault.set-contract-owner()` against his will.

Recommendation

Prefer `contract-caller` to `tx-sender` for authentication. `contract-caller` returns the caller of the current contract context.

Status

Acknowledged. A new development would address this issue.

ME-02 Independent Winning Probability in Lottery

Location:

- `clarity/contracts/pool/alex-launchpad.clar`

As it was described in CR-01, each `claim()` execution generates a new random number. Therefore, while there are tokens to transfer, the winning probability is independent and the amount of winners is unknown. In order to solve it, the function checks if all the tickets provided were won before executing the rest of the function. When all the winner tickets are determined, the listing is completed and new claims are not accepted.

This mechanism may result in two issues. Firstly, some tokens might not be claimed if there are not enough winner tickets. Furthermore, there is no use given to this remainder tokens. Secondly, it generates a race condition between the users to claim before the listing is completed. Otherwise, a user will not be able to claim even when he has tickets.

Recommendation

A solution would be to generate only n random numbers, where n is the number of winning tickets. Then, `claim()` would check if one of the random numbers were in the user's range of tickets. However, due to Clarity limitations, this solution cannot be implemented.

Status
Acknowledged.

Minor Severity Issues

MI-01 Arithmetic Underflow Calculating Staking Reward

Location:

- `clarity/contracts/pool/alex-reserve-pool.clar:[379]`

In `get-entitled-staking-reward()`, the rewards are calculated with the quotient between the amount staked by the user and the total amount staked, multiplied by the token's coinbase amount.

```
(define-private (get-entitled-staking-reward (token principal) (user-id uint) (target-cycle
uint) (stacks-height uint))
 (let
  (
   (total-staked-this-cycle (get-staking-stats-at-cycle-or-default token target-cycle))
   (user-staked-this-cycle (get amount-staked (get-staker-at-cycle-or-default token
target-cycle user-id)))
  )
  (match (get-reward-cycle token stacks-height)
   current-cycle
    (mul-down (get-coinbase-amount-or-default token target-cycle) (div-down
user-staked-this-cycle total-staked-this-cycle))
    u0
  )
 )
)
```

Quotient's result may have a higher imprecision than the result obtained by multiplying first and then dividing it by the total staked.

Recommendation
For a more precise result, get the product between the coinbase amount and the amount staked by the user, and divide it by the total amount staked:

```
(div-down
  (mul-down (get-coinbase-amount-or-default token target-cycle) user-staked-this-cycle)
  total-staked-this-cycle
)
```

Status
Resolved.

MI-02 Ended Pool can be Created

Location:

- `clarity/contracts/pool/alex-launchpad.clar`

Pool creation function (`create-pool()`) does not validate the block numbers provided for the `registration-start` and `registration-end` variables. Therefore, a pool can be created without time for registration.

Recommendation

The input `registration-start` should be checked to be equal or greater than the current block number.

Status

Resolved.

Enhancements

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

Table

ID	Title	Status
EN-01	Missing Source Code Comments	Not implemented
EN-02	Unnecessary Computation to Check if the Listing Is Activated	Implemented

Details

EN-01 Missing Source Code Comments

Location:

- `clarity/contracts/pool/alex-launchpad.clar`

The launchpad contract lacks of function documentation in the source code. Comments documenting a function helps the contract reader to understand better the usage of that piece of code.

Status

Not implemented. The development team committed to add the documentation.

EN-02 Unnecessary Computation to Check if the Listing Is Activated

Location:

- `clarity/contracts/pool/alex-launchpad.clar`

The `listing` mapping contains a variable named `activated` that is initially set to `false` when the pool is created and then increased for each new register. The `register()` function updates the value to `true` if the amount of tickets validated (`total-subscribed`) reached the `activation-threshold`. There is no other function that updates the `total-subscribed` value nor the `activated` value. However, the getter function `is-listing-activated()` computes again the comparison between the two variables instead of reading from the `activated` variable.

Recommendation

Read from the `activated` variable instead of performing the comparison again.

Status

Implemented.

Changelog

- 2022-01-07 – Initial report based on commit `44c44846bfbcce6096be04bd1380728c98f09ec8`.
- 2022-01-11 – Reaudit report based on the fixes in commit `31a5d660c83d41c10cd1b34498f02bc3a407721e`.
- 2022-01-13 – CR-01 state changed to “acknowledged”.