Beetroot Finance Audit Report

Tue Feb 18 2025

contact@bitslab.xyz https://twitter.com/tonbit_

TonBit

Beetroot Finance Audit Report

1 Executive Summary

1.1 Project Information

Description	Deposit USDT and farm the best available yield across the TON ecosystem
Type	DeFi
Auditors	TonBit
Timeline	Wed Feb 05 2025 - Tue Feb 18 2025
Languages	FunC
Platform	Ton
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/Beetroot-fi/contracts
Commits	68d7a2b9c16316e65c2d46afc94a929002cb52b6 636dd8427789369514c5657233f68516f081af43

1.2 Files in Scope

The following are the SHA1 hashes of the original reviewed files.

ID	File	SHA-1 Hash
GAS1	contracts/imports/gas.fc	009fc000c4b99fa1535ea4aa67771d4ec0d0af88
POO	contracts/pool.fc	cfc85be99375256813d2e30aebddfb1b037ee2d6
OCO2	contracts/imports/op-codes.fc	8b7277a2d004abf2a6c482bf1f304ce2a4de8e60
UTI	contracts/imports/utils.fc	3bd2b5bebbf262a04e9b23b9455cbc88830da4d6

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	6	6	0
Informational	2	2	0
Minor	2	2	0
Medium	1	1	0
Major	1	1	0
Critical	0	0	0

1.4 TonBit Audit Breakdown

MoveBit aims to assess repositories for security-related issues, code quality, and compliance with specifications and best practices. Possible issues our team looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Integer overflow/underflow by bit operations
- Number of rounding errors
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting
- Unchecked CALL Return Values
- The flow of capability
- Witness Type

1.5 Methodology

The security team adopted the **"Testing and Automated Analysis"**, **"Code Review"** and **"Formal Verification"** strategy to perform a complete security test on the code in a way that is closest to the real attack. The main entrance and scope of security testing are stated in the conventions in the "Audit Objective", which can expand to contexts beyond the scope according to the actual testing needs. The main types of this security audit include:

(1) Testing and Automated Analysis

Items to check: state consistency / failure rollback / unit testing / value overflows / parameter verification / unhandled errors / boundary checking / coding specifications.

(2) Code Review

The code scope is illustrated in section 1.2.

(3) Formal Verification(Optional)

Perform formal verification for key functions with the Move Prover.

(4) Audit Process

- Carry out relevant security tests on the testnet or the mainnet;
- If there are any questions during the audit process, communicate with the code owner in time. The code owners should actively cooperate (this might include providing the latest stable source code, relevant deployment scripts or methods, transaction signature scripts, exchange docking schemes, etc.);
- The necessary information during the audit process will be well documented for both the audit team and the code owner in a timely manner.

2 Summary

This report has been commissioned by Beetroot Finance to identify any potential issues and vulnerabilities in the source code of the Beetroot Finance smart contract, as well as any contract dependencies that were not part of an officially recognized library. In this audit, we have utilized various techniques, including manual code review and static analysis, to identify potential vulnerabilities and security issues.

During the audit, we identified 6 issues of varying severity, listed below.

ID	Title	Severity	Status
POO-1	Centralization Risk	Major	Fixed
POO-2	Unprocessed Bounced Messages	Medium	Fixed
POO-3	Missing Check in Swap Fee Calculation	Minor	Fixed
POO-4	Potential Precision Loss in Calculation	Minor	Fixed
POO-5	Incorrect Comment	Informational	Fixed
POO-6	Unnecessary Computation	Informational	Fixed

3 Participant Process

Here are the relevant actors with their respective abilities within the Beetroot Finance Smart Contract :

Admin

- `unstake` : Withdraws staked assets from supported protocols.
- `withdraw` : Transfers funds from the contract to a specified wallet address.
- `upgrade_contract` : Updates the data and code of the contract.
- `update_price_data` : Updates the ROOT price and APY.

User

- `transfer_usdt` : Exchanges USDT for ROOT tokens and stakes a portion of the USDT.
- `transfer_root` : Exchanges ROOT for USDT.
- `get_price_data` : Gets current ROOT price, APY, and available USDT.

4 Findings

POO-1 Centralization Risk

Severity: Major

Status: Fixed

Code Location:

contracts/pool.fc#291-298

Descriptions:

A centralization risk was identified in the smart contract:

The current `admin` has unrestricted control over critical contract functions, including:

- Arbitrarily updating price data
- Performing `unstake` operations
- Transferring any funds
- Upgrading the contract at any time

If the `admin` account is compromised, all contract funds would be at risk.

```
if (op == op::update_price_data) {
    throw_unless(err::not_admin, equal_slices_bits(sender_address, admin));

    int new_root_price = in_msg_body~load_uint(64);
    int new_apy = in_msg_body~load_uint(32);
    in_msg_body.end_parse();

    root_price = new_root_price;
    current_apy = new_apy;
    save_data();

    return ();
}
```

Suggestion:

- **Use multisig mechanisms** to mitigate the risk of a single-point failure.

- **Introduce a time delay** for contract upgrades to allow users to react in case of unexpected changes.
- **Refer to Stonfi's implementation** for secure upgrade mechanisms: [Stonfi Admin Calls]

Resolution:

This issue has been fixed. The client has adopted our suggestions.

POO-2 Unprocessed Bounced Messages

Severity: Medium

Status: Fixed

Code Location:

contracts/pool.fc#160-187

Descriptions:

In the `receive_usdt` message handler, the contract uses part of the user's funds to **buy TLP** and **stake TON hedge** while setting the sent messages as **bounced**. This means that if the **buy** or **stake** operations fail, the bounced messages can be used to handle the failed transactions appropriately.

However, the contract **does not handle bounced messages** and simply ignores them instead of implementing recovery or refund logic.

```
cell buy_tlp_fwd_payload = begin_cell()
    .store_uint(op::tradoor_create_increase_lp_pos_order, 8)
    .store_coins(tradoor_deposit_amount)
    .store_coins(50000000) ;; 0.05 ton
    .end_cell();
send_jettons(
    tradoor_deposit_amount,
    tradoor_pool,
    query_id,
    usdt_jetton_wallet,
    buy_tlp_fwd_payload,
    100000000, ;; 0.1 ton
    150000000, ;; ;; 0.15 ton
     owner
);

;; stake ton hedge
send_jettons(
    ton_hedge_deposit_amount,
    ton_hedge_pool,
    query_id,
    usdt_jetton_wallet,
    begin_cell().store_uint(op::ton_hedge_stake, 32).end_cell(),
    110000000, ;; 0.11 ton
    150000000, ;; 0.15 ton
    owner
);
```

Suggestion:

It is recommended to **explicitly handle bounced messages** in the contract.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

POO-3 Missing Check in Swap Fee Calculation

Severity: Minor

Status: Fixed

Code Location:

contracts/pool.fc#153

Descriptions:

The line `jetton_amount -= ONE_USDT;` lacks a check for negative values. If `jetton_amount` is smaller than `ONE_USDT`, this operation could result in a negative value, leading to undefined behavior or potential vulnerabilities.

Suggestion:

To mitigate this risk, it is recommended to add an check before performing the subtraction.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

POO-4 Potential Precision Loss in Calculation

Severity: Minor

Status: Fixed

Code Location:

contracts/pool.fc#192

Descriptions:

There is a calculation for `mint_amount` that involves division followed by multiplication:

```
int mint_amount = (jetton_amount / root_price) * 100000;
```

Performing division before multiplication can result in truncation of fractional values, reducing the accuracy of the calculation.

Suggestion:

Precision loss can lead to incorrect calculations, especially when dealing with small values or high-precision requirements.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

POO-5 Incorrect Comment

Severity: Informational

Status: Fixed

Code Location:

contracts/pool.fc#160

Descriptions:

In the contract, the comment during the `buy TLP` operation is mistakenly written as `but tlp` instead of `buy tlp`.

```
;; but tlp
cell buy_tlp_fwd_payload = begin_cell()
    .store_uint(op::tradoor_create_increase_lp_pos_order, 8)
    .store_coins(tradoor_deposit_amount)
    .store_coins(50000000) ;; 0.05 ton
    .end_cell();
send_jettons(
    tradoor_deposit_amount,
    tradoor_pool,
    query_id,
    usdt_jetton_wallet,
    buy_tlp_fwd_payload,
    100000000, ;; 0.1 ton
    150000000, ;; ;; 0.15 ton
     owner
);
```

Suggestion:

Updating the comment will improve the clarity and maintainability of the code.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

POO-6 Unnecessary Computation

Severity: Informational

Status: Fixed

Code Location:

contracts/pool.fc#82-86

Descriptions:

A certain calculation is performed unconditionally, even though its result is only utilized within the `else` condition. This leads to unnecessary computation when the `if` condition is met, which often happens.

```
int int_profit_part = profit / ONE_USDT;
int frac_profit_part = profit - (int_profit_part * ONE_USDT);

slice int_profit_part_s = convert_int_to_slice(int_profit_part);
slice frac_profit_part_s = convert_int_to_slice(frac_profit_part);
```

Suggestion:

To optimize performance, the calculation should be moved inside the else block.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

Appendix 1

Issue Level

- **Informational** issues are often recommendations to improve the style of the code or to optimize code that does not affect the overall functionality.
- **Minor** issues are general suggestions relevant to best practices and readability. They don't post any direct risk. Developers are encouraged to fix them.
- **Medium** issues are non-exploitable problems and not security vulnerabilities. They should be fixed unless there is a specific reason not to.
- **Major** issues are security vulnerabilities. They put a portion of users' sensitive information at risk, and often are not directly exploitable. All major issues should be fixed.
- **Critical** issues are directly exploitable security vulnerabilities. They put users' sensitive information at risk. All critical issues should be fixed.

Issue Status

- **Fixed:** The issue has been resolved.
- **Partially Fixed:** The issue has been partially resolved.
- **Acknowledged:** The issue has been acknowledged by the code owner, and the code owner confirms it's as designed, and decides to keep it.

Appendix 2

Disclaimer