ThunderFinance Preliminary Audit Report

Wed May 01 2024

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

TonBit

ThunderFinance Preliminary Audit Report

1 Executive Summary

1.1 Project Information

Description	Thunder Finance offers a one-stop liquidity mining solution that enables any protocol to quickly establish a farming pool
Type	DeFi
Auditors	TonBit
Timeline	Thu Apr 25 2024 - Wed May 01 2024
Languages	Tact
Platform	Ton
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/Ton-Dynasty/ThunderFinance
Commits	3ff6f55f76a50c8083d218828c7f38122bd16f7a f6aa23b467bc2ff158bc385be89edae20377bc59 8c821a9f41d6ca1cd8f99a9ffe43f81b1ec729af

1.2 Files in Scope

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

ID	File	SHA-1 Hash
DAT	contracts/packages/utils/data.fc	cad660781bf6bde54f9bc42d4dd3e892a37c473a
STD	contracts/imports/stdlib.fc	2f104cd568a4cebb1c4112ecf8979800f0672575
TMC	contracts/ton_master_chef.tact	72f6626f3d90624e0f2e33bd180baa6fbc475c68
JMC	contracts/jetton_master_chef.tact	e4992d942c37c7a11e4dfbfc9b6511bee7046cd4
MCH	contracts/mini_chef.tact	fb930849417bae1683db3fcf0ea119f4af4e2dfe
EST	contracts/packages/utils/Estimatable.tact	b02f76bf1cbba6b8f2fb65618908de7d9e54d9c5
LOC	contracts/packages/utils/Lockable.tact	217ab7c9dbdc3988d7d8cad7735527c655b2341d
JMA	contracts/packages/token/jetton/JettonMaster.tact	9b124fc400f6279e47c6b6055b710b3565742718
JWA	contracts/packages/token/jetton/JettonWallet.tact	43ef2a5f8d62c56535bbb22466551dd987ab9b83
MES	contracts/messages.tact	9dafc7919f3109fe7393706328e44b78072b1428
CTMCT	contracts/trait_master_chef.tact	1beb5020774e509b4ed31cf6f877ef090c4870da

KIT	contracts/kitchen.tact	4a9c93518a378d98068086165ae3aca8058f8b47
JET	contracts/jetton.tact	b7a30c7f520c155cbc1654d7bb84703dea603dec
TMC	contracts/ton_master_chef.tact	c3c5b960ba4d4f959a71270568fd6824c43b9b76
JMC	contracts/jetton_master_chef.tact	065c11d10536b1cc49b0607324257548f84cc3a3
MCH	contracts/mini_chef.tact	ea42693af7846191c91a6fc4750a4801d421813d
JET	contracts/packages/mock/jetton.tact	aed3f67e61dd06a4dd0f69a8d128050d6ae462e8
MES	contracts/messages.tact	4afa7608d6b964df5ec8b8a6494013f87b6a8f40
TMC1	contracts/trait_master_chef.tact	df1960679247baf4d22bf76e2de1b30b4e2693fd
KIT	contracts/kitchen.tact	69563204f2950274843232e889e9a407488221f9

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	12	12	0
Informational	3	3	0
Minor	5	5	0
Medium	3	3	0
Major	0	0	0
Critical	1	1	0
Discussion	0	0	0

1.4 TonBit Audit Breakdown

TonBit 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

1.5 Methodology

The security team adopted the **"Testing and Automated Analysis"**, **"Code Review"** 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) 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 Perman Lab to identify any potential issues and vulnerabilities in the source code of the ThunderFinance 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 12 issues of varying severity, listed below.

ID	Title	Severity	Status
JMC-1	Rewards for Users not Present in `JettonMasterChef` cannot be Extracted	Medium	Fixed
JMC-2	Duplicate Code	Informational	Fixed
JMC-3	Unused `receive()` function	Informational	Fixed
MES-1	Unused Field `thunderMintJettonWallet` in Messages	Minor	Fixed
MES-2	Message Repeat Definition	Informational	Fixed
TMC-1	Logic Flaw in LP Supply Adjustment	Critical	Fixed
TMC-2	Unchecked Start and End Times	Minor	Fixed
TMC1-1	Maliciously Initialisable Contracts	Medium	Fixed
TMC1-2	Uncalculated Gas and Unprocessed Bounce	Medium	Fixed
TMC1-3	Inconsistent Handling of Contracts for Return	Minor	Fixed

TMC1-4	Mismatch of Judgement Conditions	Minor	Fixed
TMC1-5	Redundant Field `createdAt` in `JettonMasterChef` and `TonMasterChef` Contracts	Minor	Fixed

3 Participant Process

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

Owner

- The owner can send a `SetUpJettonMC` message to initialize `JettonMasterChef`
- The owner can send a `SetUpTonMC` message to initialize the `TonMasterChef` contract and transfer the awarded Ton tokens to the contract
- The owner can send a `JettonTransferNotification` message to deposit reward tokens into the `JettonMasterChef` contract
- The owner can send an `AddPool` message to the `JettonMasterChef` or `TonMasterChef` contract to add a new pool to the contract
- The owner can send a `Set` message to the `JettonMasterChef` or `TonMasterChef` contract to change the reward allocation ratio for a given `pool`

User

- Users can send a `JettonTransfer` message to send a `lpToken` to the `JettonWallet` contract corresponding to `JettonMasterChef` or `TonMasterChef` and add a `forward_ton_amount` to the message to deposit `lpTokens` to the `JettonMasterChef` or `TonMasterChef` contract with a `JettonTransferNotification` message to deposit the `lpToken` for a reward
- Users can send `Withdraw` messages to `JettonMasterChef` or `TonMasterChef` contracts to withdraw `lpToken` previously deposited
- Users can send `Harvest` messages to `JettonMasterChef` or `TonMasterChef` contracts to withdraw rewards earned during the deposit period
- Users can send `UpdatePool` messages to `JettonMasterChef` or `TonMasterChef` contracts to update the reward allocation parameters in the corresponding pool
- Users can send a `WithdrawAndHarvest` message to the `JettonMasterChef` or `TonMasterChef` contract to withdraw the deposited `lpToken` and the rewards generated during the deposit

4 Findings

JMC-1 Rewards for Users not Present in `JettonMasterChef` cannot be Extracted

Severity: Medium

Status: Fixed

Code Location:

contracts/jetton_master_chef.tact#95-105;

contracts/ton_master_chef.tact#98-104

Descriptions:

When there was a user mining in `JettonMasterChef` or `TonMasterChef`, but the user suddenly withdraws the principal and stops mining, the subsequent rewards cannot be withdrawn. For example:

1. The staking period is 10 days.
2. On the second day, the first user deposits `lpToken` into the contract and starts mining, at which point the rewards generated on the first day are transferred to the deployer.
3. If the user withdraws the principal on the third day and terminates mining, the rewards from the third to the tenth day cannot be withdrawn.

Suggestion:

It is recommended to add logic for extracting window period rewards.

Resolution:

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

JMC-2 Duplicate Code

Severity: Informational

Status: Fixed

Code Location:

contracts/jetton_master_chef.tact#104-106

Descriptions:

The first three lines of the snippet when processing the `WithdrawAndHarvestReply` message can be replaced by the `requireMiniChef` function.

```
let initCode: StateInit = self._calculateMiniChefInit(msg.sender);
        let expectedSender: Address = contractAddress(initCode);
        require(expectedSender == sender(), "unexpected sender");
```

Suggestion:

It is recommended to use the `requireMiniChef` function instead.

Resolution:

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

JMC-3 Unused `receive()` function

Severity: Informational

Status: Fixed

Code Location:

contracts/jetton_master_chef.tact#37

Descriptions:

The `receive()` function in the `JettonMasterChef` contract is not used.

Suggestion:

It is recommended that unused functions be removed.

Resolution:

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

MES-1 Unused Field `thunderMintJettonWallet` in Messages

Severity: Minor

Status: Fixed

Code Location:

contracts/messages.tact#24

Descriptions:

The `thunderMintJettonWallet` field in message `BuildJettonMasterChef` and `SetUpJettonMC` is not used in the contract.

Suggestion:

It is recommended that unused fields be deleted.

Resolution:

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

MES-2 Message Repeat Definition

Severity: Informational

Status: Fixed

Code Location:

contracts/messages.tact#4-19;

contracts/packages/token/jetton/JettonWallet.tact#10-25

Descriptions:

The `JettonTransfer` and `JettonTransferNotification` messages are defined duplicated in the `JettonWallet.tact` and `message.tact` files.

Suggestion:

It is recommended that the definition in one of the files be deleted.

Resolution:

This issue has been fixed. The `JettonWallet` contract is test code.

TMC-1 Logic Flaw in LP Supply Adjustment

Severity: Critical

Status: Fixed

Code Location:

contracts/trait_master_chef.tact#59

Descriptions:

When a user initiates a deposit request, the corresponding `lpSupply` in the pool is increased accordingly. However, there is no corresponding decrease in `lpSupply` when withdrawing. This is illogical, as `lpSupply` is used to calculate the `accRewardPerShare` in the pool. If its value is incorrect, it will affect the equity of the entire user base in the pool, leading to significant discrepancies between the expected and actual earnings for users.

If `lpSupply` is not deducted during withdrawals, each user deposit will result in `lpSupply` growing indefinitely. Since `lpSupply` is used to calculate `accRewardPerShare`, this will cause `accRewardPerShare` to decrease over time, approaching zero. Consequently, users will be unable to receive subsequent rewards.

The two screenshots below depict Sushi's code. It can be observed that Sushi directly deducts `lpSupply` during withdrawal. When obtaining `lpSupply` in the `updatePool` function, it directly retrieves the balance, thus obtaining the post-withdrawal value as well.

```
233         function withdraw(uint256 pid, uint256 amount, address to) public {
234             PoolInfo memory pool = updatePool(pid);
235             UserInfo storage user = userInfo[pid][msg.sender];
236 
237             // Effects
238             user.rewardDebt = user.rewardDebt.sub(int256(amount.mul(pool.accSushiPerShare) / ACC_SUSHI_PRECISION));
239             user.amount = user.amount.sub(amount);
240 
241             // Interactions
242             IRewarder _rewarder = rewarder[pid];
243             if (address(_rewarder) != address(0)) {
244                 _rewarder.onSushiReward(pid, msg.sender, to, 0, user.amount);
245             }
246 
247             lpToken[pid].safeTransfer(to, amount);
248 
249             emit Withdraw(msg.sender, pid, amount, to);
250         }
```

```
191         function updatePool(uint256 pid) public returns (PoolInfo memory pool) {
192             pool = poolInfo[pid];
193             if (block.number > pool.lastRewardBlock) {
194                 uint256 lpSupply = lpToken[pid].balanceOf(address(this));
195                 if (lpSupply > 0) {
196                     uint256 blocks = block.number.sub(pool.lastRewardBlock);
197                     uint256 sushiReward = blocks.mul(sushiPerBlock()).mul(pool.allocPoint) / totalAllocPoint;
198                     pool.accSushiPerShare = pool.accSushiPerShare.add((sushiReward.mul(ACC_SUSHI_PRECISION) / lpSupply).to128());
199                 }
200                 pool.lastRewardBlock = block.number.to64();
201                 poolInfo[pid] = pool;
202                 emit LogUpdatePool(pid, pool.lastRewardBlock, lpSupply, pool.accSushiPerShare);
203             }
204         }
```

Suggestion:

It is recommended that `lpSupply` be appropriately decreased when users withdraw funds.

Resolution:

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

TMC-2 Unchecked Start and End Times

Severity: Minor

Status: Fixed

Code Location:

contracts/trait_master_chef.tact#155-162

Descriptions:

The `Master_chef` contract was initialised without checking the sizes of `StartTime` and `Deadline`, which if equal could lead to a divide-by-zero error when calculating rewardPerSecond.

Suggestion:

It is recommended to check the value of `startTime` and `deadline` when initialising the contract.

Resolution:

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

TMC1-1 Maliciously Initialisable Contracts

Severity: Medium

Status: Fixed

Code Location:

contracts/ton_master_chef.tact#36-74

Descriptions:

When the `kitchen` contract receives the `BuildTonMasterChef` message to deploy the `tonMasterChef` contract and sends the `SetUpTonMC` message to initialise it, if the initialisation incoming Ton tokens are not enough then there may be a situation where the deployment succeeds but the initialisation fails, this time, if there is a malicious actor who sends the malicious `SetUpTonMC` message to the `tonMasterChef` contract that has already been successfully deployed, this will result in the parameters being maliciously configured in the contract.

Suggestion:

It is recommended that permissions are checked at initialisation time.

Resolution:

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

TMC1-2 Uncalculated Gas and Unprocessed Bounce

Severity: Medium

Status: Fixed

Code Location:

contracts/ton_master_chef.tact;

contracts/jetton_master_chef.tact;

contracts/trait_master_chef.tact

Descriptions:

Contracts in the project do not perform calculations about gas consumption in the contract when performing operations such as `Deposit` , `Withdraw` , `Harvest` , etc., and do not perform bounce processing in any of the `JettonMasterChef` and `TonMasterChef` related contracts, which may lead to inconsistency in the state of the contract.

Example:

1. User A prepares for a `Deposit` operation and transfers the `lpToken` to the wallet corresponding to the `lpToken` in `JettonMasterChef` .
2. The `JettonMasterChef` contract receives the `JettonTransferNotification` message and calls the internal `userDeposit` function.
3. The function internally increases the value of `pool.lpSupply` and sends a `UserDeposit` message to the user's `MiniChef` contract.
4. But the value passed in by the user is not enough to perform all the logic in `MiniChef` , at this point `MiniChef` throws an exception and does not record the number of `lpTokens` deposited by the user, but `JettonMasterChef` has already incremented the `lp.totalSupply` and has not sent the `UserDeposit` message to the user's `MiniChef` contract `totalSupply` and has not returned the user's deposited `lpToken` to the user.

Suggestion:

It is recommended to add logic to the contract that handles `bounce` and calculates whether the current `value` is sufficient to cover the `gas` required to execute the contract.

Resolution:

This issue has been fixed. The client added a constraint requiring a minimum gas threshold for user deposit operations.

TMC1-3 Inconsistent Handling of Contracts for Return

Severity: Minor

Status: Fixed

Code Location:

contracts/ton_master_chef.tact#40,48-53,60

Descriptions:

The contract handles `SetUpTonMC` messages by throwing an exception for exceptions that have already been initialised in line 40, by destroying the contract for exceptions where the `rewardPersecond` value is less than zero in line 48, and by returning directly to the contract for exceptions where the number of Ton's passed in does not match the parameters in line 60.

Suggestion:

It is recommended that the treatment of the three exceptions be consistent.

Resolution:

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

TMC1-4 Mismatch of Judgement Conditions

Severity: Minor

Status: Fixed

Code Location:

contracts/ton_master_chef.tact#60;

contracts/jetton_master_chef.tact#74

Descriptions:

The if logic judgment conditions for `JettonTransferNotification` and `SetUpTonMC` are inconsistent when adding reward tokens to a contract.

```
if(msg.amount < expectedAmount || now() > self.deadline) {
```

```
if (remainTon < expectedTon) {
```

Suggestion:

It is recommended that the judgment conditions be changed to be consistent.

Resolution:

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

TMC1-5 Redundant Field `createdAt` in `JettonMasterChef` and `TonMasterChef` Contracts

Severity: Minor

Status: Fixed

Code Location:

contracts/ton_master_chef.tact#13;

contracts/jetton_master_chef.tact#13;

contracts/trait_master_chef.tact#21

Descriptions:

In the files `jetton_master_chef.tact` and `ton_master_chef.tact`, there exists a field called `createdAt`, which is initialized to record the contract creation time but is not utilized within the contract. Additionally, the functions `getJettonMasterChefData()` and `getTonMasterChefData()` do not return this field, rendering it unnecessary in the contract.

```
createdAt: Int = 0;
```

Suggestion:

It is recommended to remove the redundant field `createdAt` from the contract files `jetton_master_chef.tact` and `ton_master_chef.tact` since it is not utilized within the contract and is not returned by any exposed functions.

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

This report is based on the scope of materials and documents provided, with a limited review at the time provided. Results may not be complete and do not include all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your own risk. A report does not imply an endorsement of any particular project or team, nor does it guarantee its security. These reports should not be relied upon in any way by any third party, including for the purpose of making any decision to buy or sell products, services, or any other assets. TO THE FULLEST EXTENT PERMITTED BY LAW, WE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, IN CONNECTION WITH THIS REPORT, ITS CONTENT, RELATED SERVICES AND PRODUCTS, AND YOUR USE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NOT INFRINGEMENT.