Lucky Train Audit Report

Tue May 27 2025

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

TonBit

Lucky Train Audit Report

1 Executive Summary

1.1 Project Information

Description	Lucky Train is a Web3 project where a staking-like mechanism is presented as an engaging train journey within a Telegram Mini App.
Type	GameFi
Auditors	TonBit
Timeline	Tue May 13 2025 - Fri May 16 2025
Languages	FunC
Platform	Ton
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/scalebit/ClientProjects2
Commits	96e9778a57aabc64f274b69b55c57d7a1da3cf0c

1.2 Files in Scope

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

ID	File	SHA-1 Hash
PAR	smart-contract-dev/contracts/jtnimports/params.fc	3e86ce82bee70992c9b0f7b4fcacf0cacfcfec1b
STD	smart-contract-dev/contracts/jtnimports/stdlib.fc	2f104cd568a4cebb1c4112ecf8979800f0672575
OCO	smart-contract-dev/contracts/jtnimports/op-codes.fc	de6e2645c68d08535a353fa1b6bde7ac915d8ef5
DPA	smart-contract-dev/contracts/jtnimports/discovery-params.fc	6809258270f1565706bba2eb7f3bcf5b2289e0ac
UTI	smart-contract-dev/contracts/jtnimports/utils.fc	19cd144cd1353e5179c9cefdd1e9b4f484f4b016
CON	smart-contract-dev/contracts/jtnimports/constants.fc	46306563a259560d0f4971082975 4698357f4d1
JUT	smart-contract-dev/contracts/jtnimports/jetton-utils.fc	e725b3a317c7c347307c6c7a4b689119c04c8b58
STD1	smart-contract-dev/contracts/imports/stdlib.fc	0c324bdf98718d711e5b28d34d6fe319e1197732
MES	smart-contract-dev/contracts/imports/messages.fc	15385fe385fdc0fa91f9fa5f59a75bb1fda2fe46
UTI1	smart-contract-dev/contracts/imports/utils.fc	fbd7a21c1958f9c4ba139aaac374ccec96e93dc5

CON1	smart-contract-dev/contracts/imports/constants.fc	8dfb5e8132502a60bdb55eef01a3a65fc22795e7
STA	smart-contract-dev/contracts/staking.fc	7fd19dfb65d50957551aacbab7e25f677579b1ba
TIC	smart-contract-dev/contracts/ticket.fc	c7b75fc826c94a478291c51f0200f3f40fa5a059
X2T	smart-contract-dev/contracts/xbrc20token.fc	b3c476797f4c9d246fe3d39b1bd7ee14adaf57e4

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	8	8	0
Informational	2	2	0
Minor	2	2	0
Medium	2	2	0
Major	2	2	0
Critical	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 Lucky Train to identify any potential issues and vulnerabilities in the source code of the Lucky Train 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 8 issues of varying severity, listed below.

ID	Title	Severity	Status
STA-1	Centralization Risk	Major	Fixed
STA-2	Incorrect During Verification	Medium	Fixed
STA-3	Not Strictly Checked	Minor	Fixed
STA-4	Redundant Code	Informational	Fixed
STA-5	Ensure All Ticket Holders Can Receive Rewards	Informational	Fixed
TIC-1	Using `op::forceClaim` Can Bypass Staking Duration Limit	Major	Fixed
TIC-2	Inconsistent Time Calculation	Medium	Fixed
TIC-3	`op::forceClaim` Causes Ticket Destruction	Minor	Fixed

3 Participant Process

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

Admin

- Can pause ticket purchases
- Can withdraw funds from the reward pool
- Can withdraw funds from the `teamBalance`
- Can send logs
- Can add and remove managers
- Can change the admin
- Can update contract code and data
- Can update distribution configuration (team/burn percentages)

Manager

- Can update distribution configuration (team/burn percentages)

User

- Can purchase tickets
- Can stake funds
- Can redeem funds and receive rewards

4 Findings

STA-1 Centralization Risk

Severity: Major

Status: Fixed

Code Location:

smart-contract-dev/contracts/staking.fc#262

Descriptions:

The owner of the `staking` contract can withdraw all funds from the reward pool.

Suggestion:

Set the owner of the `staking` contract to a multisig account.

Resolution:

This functionality was implemented intentionally and does not pose a centralization risk. User funds are not accumulated in a shared pool but are instead locked in individual smart contracts, which are automatically created upon ticket purchase. Upon completion of the staking period, these funds are returned directly to the user from their personal contract. The reward pool, from which users receive additional payouts, is funded by the administration. Therefore, the ability to both deposit to and withdraw from the pool was originally included to allow for adjustments in case of surplus accumulation.
However, in the interest of greater transparency and building user trust, we have decided to completely remove the ability to withdraw funds from the reward pool on the administrator's side.

STA-2 Incorrect During Verification

Severity: Medium

Status: Fixed

Code Location:

smart-contract-dev/contracts/staking.fc#149-156

Descriptions:

```
if (ctx::jettonOp == op::buyTicket | storage::isPaused) {
        ;; Ensure enough TON for gas fees
        if (msg_value < 45000000) {
```

The correct logic here should be that if the contract is suspended, you cannot buy ticket, but here it is possible to buyTicket if the contract is suspended.

Suggestion:

It is recommended that `(ctx::jettonOp == op::buyTicket) & (~ storage::isPaused)` .

Resolution:

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

STA-3 Not Strictly Checked

Severity: Minor

Status: Fixed

Code Location:

smart-contract-dev/contracts/staking.fc#354-380,177-180

Descriptions:

```
int buyTeamPercent = in_msg_body~load_uint(7);  ;; New team percentage
int buyBurnPercent = in_msg_body~load_uint(7);  ;; New burn percentage
```

and

```
int rewardPercent = ticketDataSlice~load_uint(7);
int burnStakePercent = ticketDataSlice~load_uint(7);
```

Because the maximum value of `load_uint(7)` is 127, the parameter percentage should not exceed 100%.

Suggestion:

It is recommended that you check if the percentage is less than 100%.

Resolution:

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

STA-4 Redundant Code

Severity: Informational

Status: Fixed

Code Location:

smart-contract-dev/contracts/staking.fc#185-189

Descriptions:

```
ifnot (amount >= price) {
            ;; TODO: user sended not enough jettons to pay for ticket. We need to return
jettons if msg have enough to cover gas fees.
            throw_unless(err::staking::notEnoughJettons, amount >= price);
            ;; return ();
        }
```

`throw_unless(err::staking::notEnoughJettons, amount >= price);` This can check if `amount>price` .

Suggestion:

It is recommended that use `throw_unless(err::staking::notEnoughJettons, amount >= price);`

Resolution:

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

STA-5 Ensure All Ticket Holders Can Receive Rewards

Severity: Informational

Status: Fixed

Code Location:

smart-contract-dev/contracts/staking.fc#425

Descriptions:

1. When a user purchases a ticket, part of the funds are deposited into the reward pool. After staking, rewards are distributed proportionally.
2. To ensure that all ticket buyers receive rewards, the following condition must be met: `(ticket.price - muldiv(ticket.price, buyBurnPercent, 100) - muldiv(ticket.price, buyTeamPercent, 100)) >= muldiv(ticket.maxStake, ticket.rewardPercent, 100)`
3. Of course, if a third party injects funds into the reward pool via `op::deposit`, this condition can be bypassed.

Suggestion:

Check parameter consistency in the `op::setTicketParams` operation.

Resolution:

Here we rely on Deposit. Initially, after a deployment, the service will make a deposit of tokens to the contract, and will do so in case the tokens run out.

TIC-1 Using `op::forceClaim` Can Bypass Staking Duration Limit

Severity: Major

Status: Fixed

Code Location:

smart-contract-dev/contracts/ticket.fc#291-305

Descriptions:

In the `ticket` contract, the reward obtained by calling `op::forceClaim` is the same as that from calling `op::claim`, but the `op::forceClaim` operation does not enforce any staking duration limit.

Suggestion:

Set a lower reward rate for the `op::forceClaim` operation.

Resolution:

The `op::forceClaim` operation has been removed from the contract.

TIC-2 Inconsistent Time Calculation

Severity: Medium

Status: Fixed

Code Location:

smart-contract-dev/contracts/ticket.fc#291-305

Descriptions:

In the `op::op::forceClaim` message, `int stakeEnd = storage::timeStarted + 86400 * durationDays;` , but in `op::claim` , `int stakeEnd = storage::timeStarted + durationDays; is calculated` , the time is not consistent.

Suggestion:

It is recommended that unified time calculation.

Resolution:

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

TIC-3 `op::forceClaim` Causes Ticket Destruction

Severity: Minor

Status: Fixed

Code Location:

smart-contract-dev/contracts/ticket.fc#291

Descriptions:

1. Users incur a cost when purchasing a ticket.
2. Before the user performs staking via `op::startStake`, they can call `op::forceClaim` to destroy the ticket contract object, causing potential loss.

Suggestion:

Disallow the `op::forceClaim` call if the user has not yet staked.

Resolution:

The `op::forceClaim` operation has been removed from the contract.

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.