Ton Staking Audit Report

Fri Sep 27 2024

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

TonBit

Ton Staking Audit Report

1 Executive Summary

1.1 Project Information

Description	A TON stake protocol
Type	DeFi
Auditors	TonBit
Timeline	Tue Sep 10 2024 - Fri Sep 27 2024
Languages	Tact
Platform	Ton
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/CyberChargeOrg/TonStakingContract.git
Commits	03b9327321c581fff045642d29f6ebdf6bba5479

1.2 Files in Scope

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

ID	File	SHA-1 Hash
ST2	stakeTon2.tact	3cd8775c8c141466b34235f9f3fe35ebb9fbdcb8

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	3	0	3
Informational	0	0	0
Minor	1	0	1
Medium	2	0	2
Major	0	0	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 Cyber Charge to identify any potential issues and vulnerabilities in the source code of the StakeTon2 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 3 issues of varying severity, listed below.

ID	Title	Severity	Status
ST2-1	Missing Pause Function	Medium	Acknowledged
ST2-2	Single-step Ownership Transfer Can be Dangerous	Medium	Acknowledged
ST2-3	Lack of Parameter Validation	Minor	Acknowledged

3 Participant Process

Here are the relevant actors with their respective abilities within the StakeTon2 Smart Contract :

Owner

- The owner can start staking through the `MsgSetStart` message.
- The owner can set `minTonForStorage` and `gasConsumption` through the `MsgGasSet` message.
- The owner can set `minTonForStorage` and `gasConsumption` for subcontracts through the `MsgGasSetSub` message.
- The owner can withdraw remaining TON from the contract through the `delThis` message.
- The owner can withdraw remaining TON from the subcontracts through the `MsgStakeAddrDel` message.
- The owner can withdraw excess TON through the `MsgStakeAddrWithdraw` message.

User

- Users can stake TON through the `MsgStake` message.
- Users can withdraw TON through the `MsgWithdraw` message.

4 Findings

ST2-1 Missing Pause Function

Severity: Medium

Status: Acknowledged

Code Location:

stakeTon2.tact#79

Descriptions:

The contract contains multiple pause checks, such as `require(!self.bPause, "paused");` however, it lacks the pause functionality.

Suggestion:

It is recommended to add pause functionality.

ST2-2 Single-step Ownership Transfer Can be Dangerous

Severity: Medium

Status: Acknowledged

Code Location:

stakeTon2.tact#2

Descriptions:

Single-step ownership transfer means that if a wrong address was passed when transferring ownership or admin rights it can mean that role is lost forever. If the admin permissions are given to the wrong address within this function, it will cause irreparable damage to the contract. Below is the official documentation explanation from OpenZeppelin

https://docs.openzeppelin.com/contracts/4.x/api/access

Ownable is a simpler mechanism with a single owner "role" that can be assigned to a single account. This simpler mechanism can be useful for quick tests but projects with production concerns are likely to outgrow it.

The `stakeTon2` contract references the ownable library, which has a single-step ownership transfer process. This is quite risky.

```
import "@stdlib/ownable";
```

https://github.com/ton-core/tact/blob/main/stdlib/libs/ownable.tact

Suggestion:

It is recommended to use a two-step ownership transfer pattern.

ST2-3 Lack of Parameter Validation

Severity: Minor

Status: Acknowledged

Code Location:

stakeTon2.tact#238-249

Descriptions:

The `MsgSetStart` message is used to set the start and end times for staking, but it lacks parameter validation. Specifically, it is essential to assert that the end time is greater than the start time to prevent logical errors or misuse.

```
receive(msg: MsgSetStart) {
    self.requireOwner();
    if (msg.bStart == 1){
        self.bStart = true;
    } else {
        self.bStart = false;
    }
    self.startTime = msg.startTime;
    self.endTime = msg.endTime;
    if (self.endTime <= self.startTime) {
        revert("End time must be greater than start time.");
    }
    self.rebackTon(sender());
}
```

Suggestion:

It is recommended to implement parameter validation checks.

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