Security Audit

By

BCS Security

Customer: Agora

Audit Date: 22nd March 2022

INTRODUCTION:

The audit document highlights the syntax, standards, semantics, and security of smart contracts. The report ensures manual and tool-based proper assessment of smart contract code. The **BCS Security** team started with analyzing and understanding contract architecture and code design patterns. The following focus is on security flaws, quality, and correctness. Also, performed line by line manual analyses for the potential issue.

AUDITING METHODOLOGY:

We perform the audit according to the following procedure:

• **Basic Coding Bugs:** We statically analyze given smart contracts for known coding bugs, and then manually verify (reject or confirm) all the issues.

• **Semantic Consistency Checks:** We then manually check the logic of implemented smart contracts.

• **Advanced Security Checks:** We further review business logic, examine system operations to uncover possible pitfalls and/or bugs.

• **Additional Recommendations:** We also provide additional suggestions regarding the coding and development of smart contracts from the perspective of proven programming practices.

CRITICAL ISSUES (critical, high severity):

Bugs and vulnerabilities that enable theft of funds, lock access to funds without possibility to restore it, or lead to any other loss of funds to be transferred to any party; data manipulations; high priority unacceptable bugs for deployment at mainnet;

ERRORS, BUGS AND WARNINGS (medium, low severity):

Bugs that can trigger a contract failure, with further recovery only possible through manual modification of the contract state or contract replacement altogether; Lack of necessary security precautions;

OPTIMIZATION POSSIBILITIES (very low severity):

Possibilities to decrease the cost of transactions and data storage of Smart-Contracts.

NOTES AND RECOMMENDATIONS (very low severity):

Tips and tricks, all other issues and recommendations, as well as errors that do not affect the functionality of the Smart-Contract.

Audit Summary:

In the Smart-Contracts were found one high severity critical issues. The code was manually reviewed for all commonly known and more specific vulnerabilities. Security engineers found 3 medium severity, 2 low severity and 4 informational issues during the audit.

SOURCE:

Files:

1. GAOERC20.sol
2. AgoraToken.sol
3. Staking.sol

Low severity issues:

1. **Public function that could be declared external**: Public functions like mint, burn, stake and unstake that are never called by the contract should be declared external to save gas.

   Source: AgoraToken.sol, Staking.sol

2. **Variable Packing could be optimized:** Struct StakingPositionFrontend uses five memory slots. The variables used inside struct can be rearranged to fit them into four slots.

   Suggestion:

   Current structure:

```
struct StakingPositionFrontend {
        uint256 amount;
        uint32 time;
        uint32 lastWithdrawal;
        uint32 createdAt;
        uint256 unclaimedRewards;
        uint256 dailyRewards;
        uint32 withdrawIn;
}
```

   Suggested structure:

```
struct StakingPositionFrontend {
        uint256 amount;
        uint32 time;
        uint32 lastWithdrawal;
        uint32 createdAt;
        uint32 withdrawIn;
        uint256 unclaimedRewards;
        uint256 dailyRewards;
}
```

Source: staking.sol

Agora team response: The issue was recognized and fixed

Medium severity issues:

1. **Variable managed improperly:** Variable named stakingPositionsByAddress is managed improperly inside function unstack. All data related to position made zero but it still occupies memory as position is neither removed from array nor the array length is reduced making StakingPosition incremental only in nature. This will also make view function stakingPositions inoperable for larger array lengths.

   Suggestion: To prevent gap in given address' StakingPosition array, last position should be stored at index of position to delete, and then delete the last slot (swap and pop).

   Source: Staking.sol

2. **Invalid function call:** token.transferFrom will be failed if staking contract is not globally approved.

   Suggestion: Use token.transfer function to transfer tokens from contract to given address.

   Source: Staking.sol

High severity issues:

1. **Reward loss:** Users will lose rewards if they unstake before claiming rewards as position details will be unset at time of unstaking.

   Suggestion: Transfer unclaimed rewards by calling claimRewards function inside function unstake before clearing position details.

   Source: Staking.sol

   Agora team response: The issue was recognized and fixed

Informational issues:

1. **Add viewability:** Consider adding view function to check if the given address is _globallyApprovedSpenders or not.

   Source: GAOERC20.sol

2. **Add traceability:** Emit an event when _globallyApprovedSpenders gets updated.

   Source: GAOERC20.sol

3. **Use NatSpec properly:** Add NatSpec for the functions with @dev, @notice, @param etc. tag for providing better definition of the function.

Source: Staking.sol

4. **Add sanity check:** Add validation to check reward amount is non zero (i.e.require(rewards > 0)) inside function claimRewards in order to avoid zero token minting.

   Source: Staking.sol