AGC Smart Contract Audit Report

Taha Karim
taha@tephracore.com

August 25, 2024

Contents

1 Introduction

This report presents a security audit of the smart contract code of the $AGC token on Polygon with the following address: 0x2Ad2934d5BFB7912304754479Dd1f096D5C807Da The audit focuses on identifying potential vulnerabilities, evaluating the overall security, and recommending improvements.

2 Contract Overview

The smart contract under review is an ERC-20 token contract with additional functionality to lock and unlock tokens. The smart contract address is Timestamp of the analysis:

The main features include:

- Standard ERC-20 functionality (minting, transferring, etc.).
- Locking and unlocking of tokens by transferring them to and from the contract.
- Ownership management through the `Ownable` contract.

3 Code Review

3.1 Lock and Unlock Functionality

The smart contract implements the following functions for locking and unlocking tokens:

```
function lock(address account, uint256 amount) external {
    _transfer(msg.sender, address(this), amount);
    emit Locked(account, amount);
}
```

Listing 1: Lock Function

```
function unlock(address account, uint256 amount) external {
    require(balanceOf(address(this)) >= amount, "Not enough locked tokens");
    _transfer(address(this), account, amount);
    emit Unlocked(account, amount);
}
```

Listing 2: Unlock Function

3.2 Identified Issues

3.2.1 Issue 1: Lack of Access Control in `unlock` Function

Description: The `unlock` function lacks proper access control, allowing any user to unlock tokens from the contract, regardless of whether they were the ones who locked them.

Impact: This could result in unauthorized users withdrawing tokens that do not belong to them, leading to potential loss of funds.

Recommendation: Implement access control by keeping track of locked balances for each user and only allowing users to unlock the tokens they have locked.

```
mapping(address => uint256) private _lockedBalances;

function unlock(uint256 amount) external {
    require(_lockedBalances[msg.sender] >= amount, "Not enough locked tokens");
    _lockedBalances[msg.sender] -= amount;
    _transfer(address(this), msg.sender, amount);
    emit Unlocked(msg.sender, amount);
}
```

Listing 3: Improved Unlock Function

3.2.2 Issue 2: No Tracking of Locked Balances

Description: The original implementation does not track the balances of locked tokens for each user. This means the contract only knows the total amount of tokens it holds, not which user locked how many tokens.

Impact: Without tracking locked balances per user, it is impossible to ensure that a user is only unlocking their own tokens, leading to security vulnerabilities.

Recommendation: Introduce a mapping to track how many tokens each user has locked. Update this mapping in both the `lock` and `unlock` functions.

```
function lock(uint256 amount) external {
    _transfer(msg.sender, address(this), amount);
    _lockedBalances[msg.sender] += amount;
    emit Locked(msg.sender, amount);
}
```

Listing 4: Improved Lock Function

4 General Recommendations

4.1 Add Event Emission for Critical Operations

To improve transparency and off-chain tracking, ensure that all critical operations such as locking and unlocking tokens emit appropriate events. This will make it easier for external services to monitor contract activity.

4.2 Implement Role-based Access Control

For future improvements, consider implementing role-based access control using the OpenZeppelin `AccessControl` contract. This would allow for more granular permissions management, such as designating specific addresses that are allowed to unlock tokens.

5 Conclusion

The smart contract demonstrates a solid implementation of ERC-20 functionality with additional locking mechanisms. However, the identified issues in the lock and unlock functions represent significant security risks. By implementing the suggested improvements, the contract's security posture can be significantly enhanced.