Smart Contract Audit

Golden Goose

By,
Shebin John

16th August 2020

Index

Reading Rainbow Tip: Some threat levels or headings might be empty if there is no vulnerability/updates/suggestions found.

Introduction

The contract audited here is currently deployed in the rinkeby testnet at the address:

https://ropsten.etherscan.io/address/0xda72d63a44b0348c7812d00f0472e32464a5845b#code

The contract is a custom version of the ERC20 smart contract with some additional functions built into the system.

Based on the audit, we were able to find 2 High threats, 0 Medium threats, and 1 Low threat with some other changes in the optimization, readability, typos, and comments section.

High Threats

1. L251, if the locked amount of token in an address is higher than the balance of unlocked token in an address, then the user can bypass this modifier due to underflow. The _transfer function checks for the balance again, which solves this problem of allowing sending the locked token, but keeping this vulnerability is not advised. The use of SafeMath is recommended.
2. L346, function transferByOwner() documentation says contradicting things. Like “Owner should send locked tokens only once to users” at the same time, it is written later “If more locked tokens will be tried to send to same address, token will be added to locked tokens and time will be replaced by the new locking time for all tokens”. While in the smart contract, L186, the locked amount is overwritten rather than added to the total. This can result in the owner forfeiting the previous gains and/or reducing the previous gains while adding additional tokens. Either SafeMath add is recommended, or allowing only calling this contract by the owner once for a particular address is recommended.

Medium Threats

None

Low Threats

1. L219, underflow problem can result in setting a time which can never be reached. Having SafeMath, using it, would be better than to think there can be no human error from the Owner. The same can be seen in L230, instead of underflow, here we can have an overflow, resulting in getting the tokens faster than intended. Another one if L185.

Optimization & Readability

1. L152, function transferOwnership() can include the @param Natspec to specify the “newOwner”. The same problem can be seen in L166, function setAllTransfersLockStatus(), L184 function addLockingTime, etc.
2. L152, function transferOwnership(), as it is a crucial function, it would be apt to include an event that specifies that the owner has been changed from the old one to a new one, with the old and new address of owners as the parameter of the event.
3. L152, function transferOwnership() can be made external instead of public, as there are no other functions that are using this function. Same can be said for L176 function getAllTransfersLockStatus(), L195 function checkLockingTimeByAddress(), etc.
4. L251, using “(_balances[_address].sub(_lockedAmount[_address]) >= requestedAmount)” is much safer and readable and still uses only two logical operations.
5. L267 function name() can include the @return Natspec to specify the “name” of the Token. The same can be seen in L274 function symbol(), L281 function decimals(), etc.
6. L376, no need to increment the airdropcount each time in the loop. Instead, could write the final count after the loop, saving a lot of gas.

Typo’s & Comments

1. L137, the comment can be updated as “You are not authenticate**d** to make this transfer.”
2. L341, “@dev Transfer tokens to a s**p**ecified address (For Only Owner)”
3. L355, “@param to address to ~~be~~ transfer tokens.”

Suggestions

1. It is unclear what happens when now is equal to time{address]. Using “<=” in L250 may reduce this ambiguity. It is for a second, but there can be instances of that happening, however small the changes can be.
2. L253 to L255 is not required. As in any case, if it enters the else clause, the result is true in the require statement at L254. Can delete the entire else block.
3. L100, the variable isValue is not supposed to be used anywhere.