QuillAudits

Audit Report

August, 2020

GOLDEN GOOSE®

Contents

Introduction

This Audit Report highlights the overall security of GoldenGoose Smart Contract. With this report, we have tried to ensure the reliability of their smart contract by complete assessment of their system's architecture and the smart contract codebase.

Auditing Approach and Methodologies applied

The Quillhash team has performed thorough testing of the project starting with analysing the code design patterns in which we reviewed the smart contract architecture to ensure it is structured and safe use of third party smart contracts and libraries.

Our team then performed a formal line by line inspection of the Smart Contract to find any potential issue like race conditions, transaction-ordering dependence, timestamp dependence, and denial of service attacks.

In the Unit testing Phase, we coded/conducted Custom unit tests written for each function in the contract to verify that each function works as expected. In Automated Testing, We tested the Smart Contract with our in-house developed tools to identify vulnerabilities and security flaws.

The code was tested in collaboration of our multiple team members and this included -

- Testing the functionality of the Smart Contract to determine proper logic has been followed throughout the process.
- Analysing the complexity of the code by thorough, manual review of the code, line-by-line.
- Deploying the code on testnet using multiple clients to run live tests
- Analysing failure preparations to check how the Smart Contract performs in case of bugs and vulnerabilities.
- Checking whether all the libraries used in the code are on the latest version.
- Analysing the security of the on-chain data.

Audit Details

Project Name: Golden Goose
Website/Etherscan Code: Etherscan
Languages: Solidity (Smart contract), Javascript (Unit Testing)
Platforms and Tools: Remix IDE, Truffle, Truffle Team, Ganache, Slither, Surya

Summary of Golden Goose Smart Contract

QuillAudits conducted a security audit of a smart contract of Golden Goose. Golden Goose contract is used to create the ERC20 token, which is a GOLD TOKEN, Smart contract contains basic functionalities of an ERC20 token.

Name: Golden Goose
Symbol: GOLD
Total supply : 500,000,000 GOLD

And some advanced features other than essential functions.

- Locking of tokens
- Locking can be done by the owner only
- Airdrop multiple addresses
- The owner can Transfer locked tokens of users
- With basic get the function to retrieve data.

Audit Goals

The focus of the audit was to verify that the smart contract system is secure, resilient and working according to its specifications. The audit activities can be grouped in the following three categories:

Security
Identifying security related issues within each contract and the system of contracts.

Sound Architecture
Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices.

Code Correctness and Quality
A full review of the contract source code. The primary areas of focus include:

- Correctness
- Readability
- Sections of code with high complexity
- Quantity and quality of test coverage

Security

Every issue in this report was assigned a severity level from the following:

High severity issues
They will bring problems and should be fixed.

Medium severity issues
They could potentially bring problems and should eventually be fixed.

Low severity issues
They are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

Number of issues per severity

	Low	Medium	High
Open	0	0	0
Closed	1	0	1

High severity issues

1. 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 are tried to send to the same address, the 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.

Recommedation

Either SafeMath add is recommended, or allowing only calling this contract by the owner once for a particular address is recommended.

If using Safemath:
_lockedAmount[lockingAddress] =
_lockedAmount[lockingAddress].add(amount);

Status: Fixed

Medium Severity Issues

No Medium severity issues

Low Severity Issues

1. L219, underflow problems 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.

Recommedation

The use of SafeMath in math operations is advised.
time[_affectiveAddress] =
time[_affectiveAddress].sub((_decreasedTime * 1 days));

Status: Developer will handle the issue

Unit Testing

Test Suite

Contract: Golden Goose Token Contracts

- Should correctly initialise constructor of GoldenGoose token Contract (136ms)
- Should check the name of a token (43ms)
- Should check a symbol of a token
- Should check a decimal of a token
- Should check an owner of a token
- Should check the total supply of a token contract
- Should check the status of the token lock
- Should check airDrop count
- Should check the balance of an owner
- Should be able to transfer tokens by owner only using transfer function (73ms)
- Should check the balance of a receiver
- Should not be able to transfer locking tokens by non-owner (61ms)
- Should be able to transfer locking tokens by owner account only (79ms)
- Should check the balance of a receiver accounts[2]
- Should check locking status of an account [2]
- Should check locking status of an account [1]

- Should check locking time of an account [2]
- Should not be able to transfer locked tokens by accounts[2] (58ms)
- Should be able to transfer tokens by non-owner account those are not locked (55ms)
- Should check the balance of a receiver
- Should check the balance of a sender
- Should be able to transfer tokens by non-owner account those are not locked to account which is already holding locked tokens (54ms)
- Should check the balance of a receiver
- Should check the balance of a sender
- Should be able to transfer tokens by account[2] which tokens are not locked, i.e. 10 tokens (55ms)
- Should check the balance of a sender
- Should not be able to transfer tokens by account[2] which tokens not locked, i.e. 100 tokens after spending unlock tokens (63ms)
- Should check approval by accounts 4 to accounts 2
- Should Approve accounts[4] to spend specific tokens of accounts[2]
- Should increase Approve accounts[4] to spend specific tokens of accounts[2]
- Should decrease Approve accounts[4] to spend specific tokens of accounts[2]
- Should Approve accounts[4] to spend specific tokens of accounts[2] (102ms)

- Should not be able to transferFrom tokens which are locked (79ms)
- Should not be able to stop transfer function by non-owner account (49ms)
- Should check the status of the token lock before locking
- Should be able to stop all transfer function by owner account (40ms)
- Should check the status of the token lock after locking
- Should not be able to stop transfer function by non-owner account (48ms)
- Should check the status of the token lock after locking
- Should be able to unstop all transfer function by owner account (42ms)
- Should check the status of the token lock after locking stop
- Should check approval by accounts 1 to accounts 5
- Should Approve accounts[5] to spend specific tokens of accounts[1]
- Should Approve accounts[4] to spend specific tokens of accounts[1] (51ms)
- Should check the balance of a sender before calling transferfrom
- Should not be able to stop transfer function by non-owner account (48ms)
- Should not be able to burn tokens by non-owner account (46ms)
- Should not be able to burn tokens by non-owner account (48ms)
- Should check the balance of a sender before calling transferfrom
- Should not be able to burn tokens by non-owner account (71ms)
- Should check an owner of a token (43ms)
- Should transfer ownership (47ms)

- Should check an owner of a token after ownership transfer
- Should transfer ownership again
- Should check airDrop count before doing an airdrop
- Should airdrop tokens (59ms)
- Should not be able to airdrop tokens by non-owner account (48ms)
- Should not be able to airdrop tokens when accounts and values are not equal (54ms)
- Should be able to transfer locking tokens by owner account only (50ms)
- Should be able to increase locking time (55ms)
- Should check the locking time of accounts [6]
- Should be able to decrease locking time (39ms)
- Should check the locking time of accounts [6]

Final Result of Test

67 Passings (3s) Passed

0 Failed

Coverage Report

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	92.5	61.76	97.06	92.77	
GoldenGoose.sol	92.5	61.76	97.06	92.77	... 328,329,330
All files	92.5	61.76	97.06	92.77	

Slither Tool Result

```
INFO:Detectors:
GoldenGoose.isValue (GoldenGoose.sol#106-109) is never used in GoldenGoose (GoldenGoose.sol#96-438)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variables
INFO:Detectors:
GoldenGoose.isValue (GoldenGoose.sol#106-109) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
INFO:Detectors:
setCompleted(uint256) should be declared external:
        - Migrations.setCompleted(uint256) (Migrations.sol#15-17)
upgrade(address) should be declared external:
        - Migrations.upgrade(address) (Migrations.sol#19-22)
getowner() should be declared external:
        - GoldenGoose.getowner() (GoldenGoose.sol#140-143)
transferOwnership(address) should be declared external:
        - GoldenGoose.transferOwnership(address) (GoldenGoose.sol#159-162)
getAllTransfersLockStatus() should be declared external:
        - GoldenGoose.getAllTransfersLockStatus() (GoldenGoose.sol#182-184)
checkLockingTimeByAddress(address) should be declared external:
        - GoldenGoose.checkLockingTimeByAddress(address) (GoldenGoose.sol#193-195)
getLockingStatus(address) should be declared external:
        - GoldenGoose.getLockingStatus(address) (GoldenGoose.sol#197-204)
name() should be declared external:
        - GoldenGoose.name() (GoldenGoose.sol#274-275)
symbol() should be declared external:
        - GoldenGoose.symbol() (GoldenGoose.sol#278-279)
decimals() should be declared external:
        - GoldenGoose.decimals() (GoldenGoose.sol#280-284)
totalSupply() should be declared external:
        - GoldenGoose.totalSupply() (GoldenGoose.sol#286-288)
balanceOf(address) should be declared external:
        - GoldenGoose.balanceOf(address) (GoldenGoose.sol#294-295)
allowance(address,address) should be declared external:
        - GoldenGoose.allowance(address,address) (GoldenGoose.sol#301-302)
transfer(address,uint256) should be declared external:
        - GoldenGoose.transfer(address,uint256) (GoldenGoose.sol#313-315)
transferFrom(address,address,uint256) should be declared external:
```

```
INFO:Detectors:
GoldenGoose.constructor(string,string,uint8,uint256,address).name (GoldenGoose.sol#119) shadows:
        - GoldenGoose.name() (GoldenGoose.sol#274-275) (function)
GoldenGoose.constructor(string,string,uint8,uint256,address).symbol (GoldenGoose.sol#119) shadows:
        - GoldenGoose.symbol() (GoldenGoose.sol#278-279) (function)
GoldenGoose.constructor(string,string,uint8,uint256,address).decimals (GoldenGoose.sol#119-120) shadows:
        - GoldenGoose.decimals() (GoldenGoose.sol#280-284) (function)
GoldenGoose.constructor(string,string,uint8,uint256,address).totalSupply (GoldenGoose.sol#120-121) shadows:
        - GoldenGoose.totalSupply() (GoldenGoose.sol#286-288) (function)
        - IERC20.totalSupply() (GoldenGoose.sol#21) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
GoldenGoose.getLockingStatus(address) (GoldenGoose.sol#197-204) uses timestamp for comparisons
        Dangerous comparisons:
        - now < time[userAddress] (GoldenGoose.sol#198-201)
GoldenGoose.decreaseLockingTimeByAddress(address,uint256) (GoldenGoose.sol#212-220) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(_decreasedTime > 0 && time[_affectiveAddress] > now,Please check address status or Incorrect input) (GoldenGoos
e.sol#216-218)
GoldenGoose.increaseLockingTimeByAddress(address,uint256) (GoldenGoose.sol#227-241) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(_increasedTime > 0 && time[_affectiveAddress] > now,Please check address status or Incorrect input) (GoldenGoos
e.sol#232-235)
GoldenGoose.transferLockedTokens(address,address,uint256) (GoldenGoose.sol#349-357) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)((_lockedAmount[from] >= value) && (now < time[from]),Insufficient unlocked balance) (GoldenGoose.sol#352-354)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
GoldenGoose.AllTransfersLockStatus() (GoldenGoose.sol#246-253) compares to a boolean constant:
        -require(bool,string)(_lockStatus == false,All transactions are locked for this contract) (GoldenGoose.sol#248-253)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality
INFO:Detectors:
```

File Description table

File Name	SHA-1 Hash
GoldenGoose.sol	953c09689163512a6effaf061cbfd668bc581da9

Implementation Recommendations

- 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.
- 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.
- 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.
- L251, using “(_balances[_address].sub(_lockedAmount[_address]) >= requestedAmount)” is much safer and readable and still uses only two logical operations.
- 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.
- L376, no need to increment the airdropcount each time in the loop. Instead, It could write the final count after the loop, saving a lot of gas.

Typo's & Comments

- L137, the comment can be updated as "You are not authenticated to make this transfer."
- L341, "@dev Transfer tokens to a specified address (For the Only Owner)"
- L355, "@param to address to be transferred tokens."

Suggestions

- 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.
- L253 to L255 is not required. As in any case, if it enters the else clause, the result is true in the required statement at L254. Can delete the entire else block.
- L100, the variable isValue is not supposed to be used anywhere.
- 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.

  Recommended
  The use of SafeMath is recommended.
  require(!(_balances[_address].sub(_lockedAmount[_address])
  < requestedAmount), "Insufficient unlocked balance");

Comments

Use case of the smart contract is very well designed and Implemented. Overall, the code is written and demonstrates effective use of abstraction, separation of concerns, and modularity. Golden Goose development team demonstrated high technical capabilities, both in the design of the architecture and in the implementation.
All the bugs, suggestions and recommends has been considered by GoldenGoose team and some of the issues they will handle to their own as those issues or calls have been handle by the only owner.

GOLDEN GOOSE®

QuillAudits

448-A EnKay Square, Opposite Cyber Hub,
Gurugram, Harayana, India - 122016

audits.quillhash.com

hello@quillhash.com