SMART CONTRACT AUDIT

ZOKYO.

Dec 15th, 2021 | v. 1.0

PASS

Zokyo's Security Team has concluded that this smart contract passes security qualifications to be listed on digital asset exchanges

TECHNICAL SUMMARY

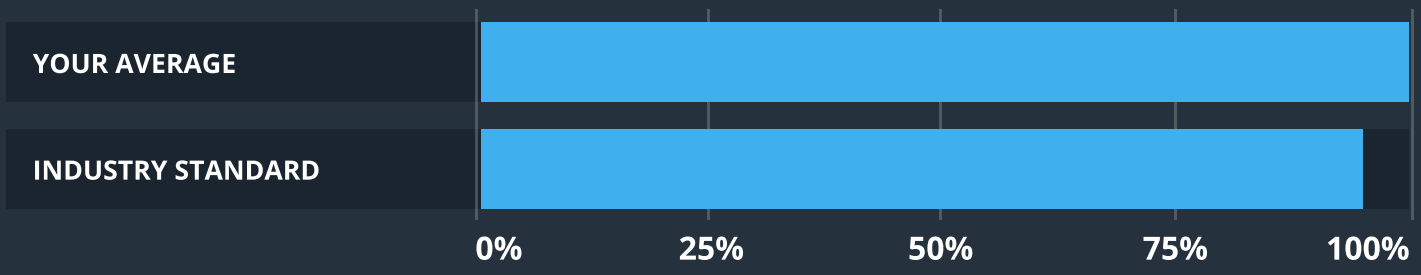
This document outlines the overall security of the Gain DAO smart contracts, evaluated by Zokyo's Blockchain Security team.

The scope of this audit was to analyze and document the Gain DAO smart contract codebase for quality, security, and correctness.

Contract Status

There were no critical issues found during the audit.

Testable Code



The testable code is 100%, which is above the industry standard of 95%.

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract, rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that's able to withstand the Ethereum network's fast-paced and rapidly changing environment, we at Zokyo recommend that the Gain DAO team put in place a bug bounty program to encourage further and active analysis of the smart contract.

TABLE OF CONTENTS

AUDITING STRATEGY AND TECHNIQUES APPLIED

The Smart contract's source code was taken from the Gain DAO repository.

Repository:
https://github.com/GainDAO/token/commit/9c5630838e1f2b6344a087ad9270cd6304901954

Last commit:
f782a0d3f03a5bf646131a7098df50c1c9f8b659

Contracts under the scope:
ERC20Distribution

Throughout the review process, care was taken to ensure that the token contract:

- Implements and adheres to existing Token standards appropriately and effectively;
- Documentation and code comments match logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices in efficient use of gas, without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the latest vulnerabilities;
- Whether the code meets best practices in code readability, etc.

Zokyo's Security Team has followed best practices and industry-standard techniques to verify the implementation of smart contracts. To do so, the code is reviewed line-by-line by our smart contract developers, documenting any issues as they are discovered. Part of this work includes writing a unit test suite using the Truffle testing framework. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

1	Due diligence in assessing the overall code quality of the codebase.	3	Testing contract logic against common and uncommon attack vectors.
2	Cross-comparison with other, similar smart contracts by industry leaders.	4	Thorough, manual review of the codebase, line-by-line.

SUMMARY

Zokyo auditing team has run a deep investigation of the Gain DAO smart contract. During the auditing process, there were found some issues. The contract is well written and structured.

It's worth mentioning that there no critical issues were found. Were found just a couple of informational issues and 1 issue with a high severity level. All of the issues were successfully resolved by the Gain DAO team.

Based on the conducted audit, we give a score of 99 to this contract.

STRUCTURE AND ORGANIZATION OF DOCUMENT

For ease of navigation, sections are arranged from most critical to least critical. Issues are tagged “Resolved” or “Unresolved” depending on whether they have been fixed or addressed. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

Critical

The issue affects the ability of the contract to compile or operate in a significant way.

High

The issue affects the ability of the contract to compile or operate in a significant way.

Medium

The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.

Low

The issue has minimal impact on the contract’s ability to operate.

Informational

The issue has no impact on the contract’s ability to operate.

MANUAL REVIEW

More logic required

HIGH | RESOLVED

The logic for KYC is missing and it is very vital to the operation of the contract. To be more specific the purchaseAllowed function doesn't have the required logic needed to perform KYC.

Recommendation:
Include the KYC logic as planned.

Logic could be improved

INFORMATIONAL | RESOLVED

Some logic in the contract can be improved to make it more logical and more readable. An example of this is the required statements starting on lines 50 and 55 respectively.
The statement require(distStartRate > 0 && distEndRate > 0, "TokenDistribution: rates should > 0") starting on line 55 should come before the statement require(distStartRate > distEndRate, "TokenDistribution: start rate should be > end rate") starting on line 50. As it is more logical and more readable to confirm the rates are greater than zero first before comparing the rates

Recommendation:
Interchange the specified lines. That is the required statement confirming the rates are greater than zero should come first and the required statement comparing the distStartRate and distEndRate should come later.

Unnecessary Function

INFORMATIONAL | RESOLVED

The function beneficiary() on line 76 is unnecessary as the public variables (_beneficiary is a public variable) come with a getter function by default and there is no need to include a getter function for them.

Recommendation:
Remove the function and use the default function _beneficiary to get the value anytime it is needed.

Redundant logic

INFORMATIONAL | RESOLVED

The logic on line 112 require(paused(), 'Distribution already started') is not needed as the WhenPaused modifier is used on the function and so the function would only run when the contract is paused.

Recommendation:
Remove the specified statement as it serves no purpose.

Redundant logic

INFORMATIONAL | RESOLVED

The logic on line 69: _kyc_approver = address(0) is not needed as the default value for variables of type address is address(0).

Recommendation:
Remove the specified statement as it serves no purpose.

Logic can be improved

INFORMATIONAL | RESOLVED

The logic in the purchaseToken can be slightly improved in order to make it more readable. The require statement on Line 197: require(actualrate>0, "unable to sell at the given rate: distribution has ended") should come before the statement at line 192: require(rate==actualrate, "unable to sell at the given rate: the rate has changed") as it is more logical and reads better to confirm the actual rate is greater than zero first before comparing the equality of the rates.

Recommendation:
Move the require statement on line 192 (the one that compares the two rates) to line 197 and then move the require statement that confirms the actual rate is greater than zero to line 192.

	ERC20Distribution
Re-entrancy	Pass
Access Management Hierarchy	Pass
Arithmetic Over/Under Flows	Pass
Unexpected Ether	Pass
Delegatecall	Pass
Default Public Visibility	Pass
Hidden Malicious Code	Pass
Entropy Illusion (Lack of Randomness)	Pass
External Contract Referencing	Pass
Short Address/ Parameter Attack	Pass
Unchecked CALL Return Values	Pass
Race Conditions / Front Running	Pass
General Denial Of Service (DOS)	Pass
Uninitialized Storage Pointers	Pass
Floating Points and Precision	Pass
Tx.Origin Authentication	Pass
Signatures Replay	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Gain DAO team

Code Coverage

The resulting code coverage (i.e., the ratio of tests-to-code) is as follows:

FILE	% STMTS	% BRANCH	% FUNCS	% LINES	UNCOVERED LINES
ERC20Distribution.sol	93.62	66.27	91.67	93.62	127,182,198
All files	**93.62**	**66.27**	**91.67**	**93.62**	

Test Results

ERC20Distribution

ERC20Distribution - KYC

- ✓ must calculate correct proof (65ms)
- ✓ the contract owner can buy tokens with no approver set (341ms)
- ✓ is able to set and update kyc approver (125ms)
- ✓ is not allowed to buy with no kyc approver set (271ms)
- ✓ is able to create and use proof with kyc approver set (652ms)

ERC20Distribution - Token distribution

- ✓ initialization - has correct beneficiary
- ✓ initialization - has correct start rate
- ✓ initialization - has correct end rate
- ✓ initialization - has correct total distribution volume
- ✓ initialization - has correct start distribution volume
- ✓ initialization - is paused after creation
- ✓ start distribution - it can receive tokens for distribution (62ms)
- ✓ start distribution - it has received the correct amount of tokens
- ✓ start distribution - distribution can be started (55ms)
- ✓ start distribution - accepts correct KYC approver (42ms)

ERC20Distribution - Slippage

```
    ✓ it is possible to buy at the current rate (101ms)
    ✓ it is possible to buy at a lower than current rate (last token) (229ms)
    ✓ it is possible to buy at a lower than current rate (70ms)
    ✓ it is not possible to buy at a higher than current rate (54ms)
  ERC20Distribution - execute buycycles
    ✓ it tests small amounts at the start of distribution (fixed token amount) (129664ms)
    ✓ it tests small amounts at the tail of the distribution (fixed token amount) (144203ms)
    ✓ it tests medium amounts across the entire distribution (fixed token amount) (669023ms)
    ✓ it tests random amounts across the entire distribution (fixed token amount) (10560ms)
    ✓ is able to execute cycle for given amounts of ether (2166ms)

  24 passing (16m)
```

Tests written by Zokyo Security team

As part of our work assisting Stargate in verifying the correctness of their contract code, our team was responsible for writing integration tests using the Truffle testing framework.

Tests were based on the functionality of the code, as well as a review of the Stargate contract requirements for details about issuance amounts and how the system handles these.

Code Coverage

The resulting code coverage (i.e., the ratio of tests-to-code) is as follows:

FILE	% STMTS	% BRANCH	% FUNCS	% LINES	UNCOVERED LINES
ERC20Distribution.sol	100.00	90.00	100.00	100.00	
All files	**100.00**	**90.00**	**100.00**	**100.00**	

Test Results

Contract: ERC20Distribution

- ✓ should be paused when deployed (251ms)
- ✓ should allow kyc approver to be set (106ms)
- ✓ needs to return the right beneficiary (77ms)
- ✓ needs to return the right start distribution (79ms)
- ✓ needs to return the right end rate distribution (81ms)
- ✓ needs to return the right total distribution balance (71ms)
- ✓ needs to return the current distributed balance (64ms)
- ✓ needs to get current rate when account is paused (64ms)
- ✓ needs to allow distribution to start correctly (113ms)
- ✓ needs to allow purchase if KYC details are correct (194ms)
- ✓ needs to get the correct current rate when the distribution has started well (95ms)
- ✓ needs to allow users to purchase tokens (167ms)
- ✓ needs to return the right current rate when distribution has ended (257ms)
- ✓ fails purchase of tokens when a wrong rate is used (163ms)
- ✓ fails purchase of tokens when pool balance is not enough (153ms)

```
✓ fails purchase of tokens if kyc is failed (154ms)
✓ prevents distribution from starting if trusted token balance and total dist balance
  don't match (88ms)
✓ doesn't deploy if the right variables are not provided (94ms)

18 passing (2s)
```

We are grateful to have been given the opportunity to work with the Gain DAO team.

The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them.

Zokyo's Security Team recommends that the Gain DAO team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

ZOKYO.