SMART CONTRACT AUDIT

ZOKYO.

Jan 19th, 2022 | v. 1.0

PASS

Zokyo's Security Team has concluded that this smart contract pass security qualifications and are fully production-ready

TECHNICAL SUMMARY

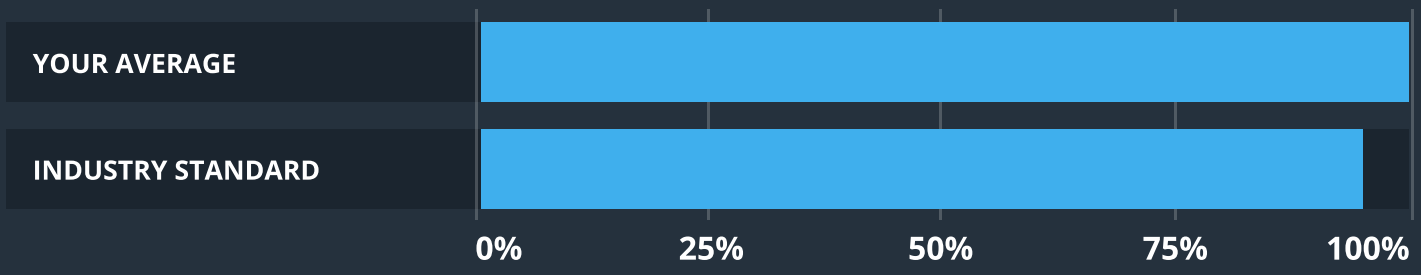
This document outlines the overall security of the Gain DAO smart contracts, evaluated by Zokyo's Blockchain Security team.

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 3

Executive Summary 5

Structure and Organization of Document 6

Manual Review 7

Code Coverage and Test Results for all files 10

- Tests written by Gain DAO team 10
- Tests written by Zokyo Secured team 13

AUDITING STRATEGY AND TECHNIQUES APPLIED

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

Repository:
https://github.com/GainDAO/token/

Last commit:
e1e9641f5f55f579a53598914afe7f6db1c5d6fa

Contracts under the scope:

- GainDAOToken;
- 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.
2	Cross-comparison with other, similar smart contracts by industry leaders.
3	Testing contract logic against common and uncommon attack vectors.
4	Thorough, manual review of the codebase, line-by-line.

SUMMARY

The Zokyo team has conducted a security audit of the given codebase. The contracts provided for an audit are well written and structured. All the findings within the auditing process are presented in this document.

During the auditing process, our auditor's team found 1 issue with a medium severity level, 1 issue with a low severity level, which were successfully resolved by the GainDao team.

Based on the results of the audit, we can give a score of 99 and state that the audited contracts are fully production-ready.

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

Possibility to transfer tokens while GainDAOToken is paused

MEDIUM | RESOLVED

According to the comments in function _beforeTokenTransfer it's allowed only to mint or burn tokens while the contract is on pause.

```
function _beforeTokenTransfer(
    address from,
    address to,
    uint256 amount
) internal override {
    // if we are not minting or burning tokens, check wether we are
    // paused or not
```

But the contract allows anyone with BURNER_ROLE to send tokens to any address. As you use AccessControlEnumerable from openzeppelin libraries, the admin can set chosen addresses to BURNER_ROLE and allow them to benefit with more permissions while the contract is on pause.

Recommendation:
Remove permission from BURNER_ROLE to transfer tokens while a contract is on pause or change comment if this is expected behavior.

Lock pragma to a specific version

LOW | RESOLVED

At the current state, pragma is set to version range ^0.8.0 and it is better to lock the pragma to a specific version since not all the EVM compiler versions support all the features, especially the latest one's which are kind of beta versions, So the intended behavior written in code might not be executed as expected.

Locking the pragma helps ensure that contracts do not accidentally get deployed using, for example, the latest compiler which may have higher risks of undiscovered bugs.

Recommendation:
Lock pragma to a specific version.

	GainDAOToken	ERC20Distribution
Re-entrancy	Pass	Pass
Access Management Hierarchy	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass
Unexpected Ether	Pass	Pass
Delegatecall	Pass	Pass
Default Public Visibility	Pass	Pass
Hidden Malicious Code	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass
External Contract Referencing	Pass	Pass
Short Address/ Parameter Attack	Pass	Pass
Unchecked CALL Return Values	Pass	Pass
Race Conditions / Front Running	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass
Floating Points and Precision	Pass	Pass
Tx.Origin Authentication	Pass	Pass
Signatures Replay	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	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
contracts\	96.94	85.94	96.15	96.94	
GainDAOToken.sol	100.00	100.00	100.00	100.00	
ERC20Distribution.sol	93.88	71.88	92.31	93.88	129, 185, 201
All files	**96.94**	**85.94**	**96.15**	**96.94**	

Test Results

```
GainDAOToken
  initialization
    ✓ has the correct name (39ms)
    ✓ has the correct symbol
    ✓ has the correct decimals
    correctly setup the roles
      ✓ deployer is admin
      ✓ deployer is pauser
      ✓ deployer is minter
      ✓ test minter role (86ms)
      ✓ test pauser role (83ms)
      ✓ test burner role (98ms)
  pausing and unpausing
    ✓ is deployed in a paused state
    ✓ non pauser cannot unpause the token (60ms)
    ✓ cannot transfer coins when the token is paused (42ms)
```

```
    ✓ non minter cannot mint coins
    ✓ minter can mint coins while the token is paused (44ms)
    ✓ pauser can unpause the token
    ✓ can transfer coins once the token is unpaused (69ms)
  minting
    ✓ minter can mint tokens
    ✓ minting does not unpause the token (38ms)
    ✓ minting does not repause an unpaused token (44ms)
    ✓ non minter cannot mint tokens
    ✓ cannot mint more tokens that the cap
  burning
    ✓ burner cannot burn tokens when paused (50ms)
    ✓ burner can burn tokens (85ms)
    ✓ non burner cannot burn tokens
    ✓ cannot burn more tokens that balance (87ms)

25 passing (5s)
```

```
ERC20Distribution
  ERC20Distribution - Roles
    ✓ removes token contract admin (96ms)
    ✓ demotes token contract owner deployer (46ms)
    ✓ demotes distribution contract owner deployer
  ERC20Distribution - KYC
    ✓ must calculate correct proof
    ✓ contract owner can buy tokens with no approver set (147ms)
    ✓ is able to set and update kyc approver from treasury account (130ms)
    ✓ is not allowed to buy with no kyc approver set (92ms)
    ✓ is able to create and use proof with kyc approver set (295ms)
  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
    ✓ start distribution - it has received the correct amount of tokens
    ✓ start distribution - distribution can be started
    ✓ start distribution - accepts correct KYC approver
  ERC20Distribution - Slippage
    ✓ it is possible to buy at the current rate (40ms)
    ✓ it is possible to buy at a lower than current rate (last token) (109ms)
    ✓ it is possible to buy at a lower than current rate
    ✓ it is not possible to buy at a higher than current rate
  ERC20Distribution - execute buycycles
    ✓ it tests small amounts at the start of distribution (fixed token amount) (52336ms)
    ✓ it tests small amounts at the tail of the distribution (fixed token amount) (51809ms)
    ✓ it tests medium amounts across the entire distribution (fixed token amount) (241586ms)
    ✓ it tests random amounts across the entire distribution (fixed token amount) (4379ms)
    ✓ is able to execute cycle for given amounts of ether (726ms)

  28 passing (6m)
```

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
contracts\	100.00	98.44	100.00	100.00	
GainDAOToken.sol	100.00	100.00	100.00	100.00	
ERC20Distribution.sol	100.00	96.88	100.00	100.00	
All files	**100.00**	**98.44**	**100.00**	**100.00**	

Test Results

```
Contract: GainDAOToken
  GainDAOToken Test Cases
    GainDAOToken Deploy Test Cases
      ✓ should deploy with correct name (142ms)
      ✓ should deploy with correct symbol (91ms)
      ✓ should deploy with correct decimals (102ms)
      ✓ should deploy paused (65ms)
      ✓ should deploy with deployer setted to admin role (215ms)
      ✓ should deploy with deployer setted to pauser role (178ms)
      ✓ should deploy with deployer setted to minter role (139ms)
      ✓ should deploy with deployer setted to burner role (194ms)
      ✓ should deploy with correct initial total supply (72ms)
    GainDAOToken Access Control Test Cases
```

✓ shouldn't set new role by not the admin (910ms)
✓ should set new roles correctly (3498ms)
✓ shouldn't revoke role by not the admin (1141ms)
✓ should revoke role correctly (2450ms)
✓ shouldn't allow to mint tokens by not the minter (306ms)
✓ shouldn't allow to burn tokens by not the burner (277ms)
✓ shouldn't unpause by not the current pauser (363ms)

GainDAOToken Paused Test Cases

✓ should allow to mint tokens when contract on pause (410ms)
✓ shouldn't allow to burn tokens when on pause (336ms)
✓ shouldn't transfer tokens when on pause (460ms)
✓ should unpause correctly (295ms)

GainDAOToken ERC20 Capped Test Cases

✓ shouldn't allow to mint tokens more than max supply (286ms)
✓ should transfer tokens correctly (343ms)
✓ shouldn't transfer tokens to the zero address (185ms)
✓ shouldn't transfer tokens if transfer amount exceed balance (212ms)
✓ should approve correctly (284ms)
✓ shouldn't approve to the zero address (241ms)
✓ should increase allowance correctly (447ms)
✓ shouldn't increase allowance for zero address (213ms)
✓ should decrease allowance correctly (496ms)
✓ shouldn't decrease allowance for zero address (227ms)
✓ shouldn't decrease allowance below zero (216ms)
✓ should transfer tokens from address correctly (489ms)
✓ shouldn't transfer tokens from address if amount exceed allowance (210ms)
✓ should burn tokens correctly (286ms)

34 passing (19s)

Contract: ERC20Distribution

✓ should be paused when deployed (271ms)
✓ should allow KYC approver to be set only by accounts with the right role (182ms)
✓ needs to return the right beneficiary (95ms)
✓ needs to return the right start distribution (83ms)
✓ needs to return the right end rate distribution (94ms)

```
✓ needs to return the right total distribution balance (71ms)
✓ needs to return the current distributed balance (76ms)
✓ needs to get current rate when account is paused (75ms)
✓ needs to allow distribution start correctly (112ms)
✓ needs to allow purchase if KYC details are correct (208ms)
✓ needs to  get correct current rate when distribution has started well (93ms)
✓ needs to allow users purchase tokens (193ms)
✓ should allow funds to be claimed by only the right beneficiary (203ms)
✓ needs to return the right current rate when distribution has ended (218ms)
✓ fails purchase of tokens when a wrong rate is used (168ms)
✓ fails purchase of tokens when pool balance is not enough (155ms)
✓ fails purchase of tokens if KYC is failed (167ms)
✓ should not care about KYC if token purchase is from default admin (133ms)
✓ prevents distribution from starting if trusted token balance and total dist balance
  don't match (87ms)
✓ doesn't deploy if the right variables are not provided (92ms)

20 passing (3s)
```

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.