

**GAIN.Ai**  
AI POWERED POOLS

SMART CONTRACT AUDIT



February 6th 2023 | v. 1.0

# Security Audit Score

**PASS**

Zokyo Security 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 smart contracts evaluated by the Zokyo Security team.

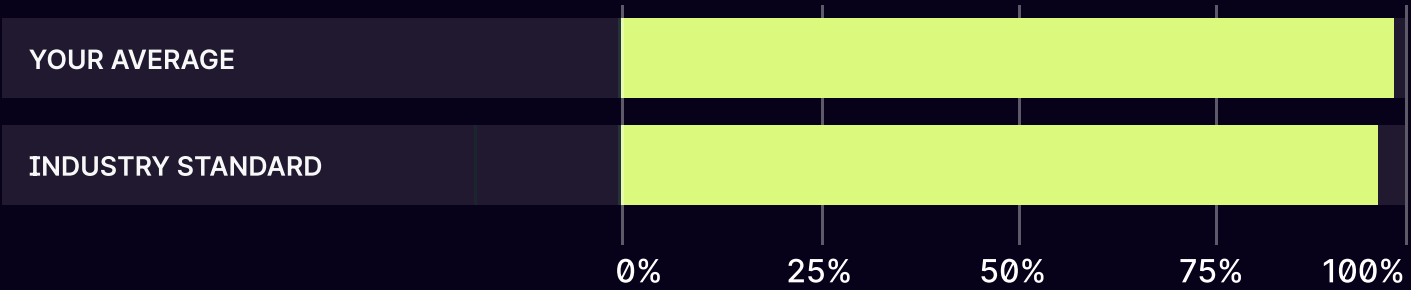
The scope of this audit was to analyze and document the Gain smart contract codebase for quality, security, and correctness.

## Contract Status



There were 0 critical issues found during the audit. (See [Complete Analysis](#))

## Testable Code



100% of the code is testable, 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 contracts but rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that can withstand the Ethereum network’s fast-paced and rapidly changing environment, we recommend that the Gain team put in place a bug bounty program to encourage further active analysis of the smart contracts.

# Table of Contents

Auditing Strategy and Techniques Applied	3
Executive Summary	4
Structure and Organization of the Document	5
Complete Analysis	6
Code Coverage and Test Results for all files written by Zokyo Security	11

# AUDITING STRATEGY AND TECHNIQUES APPLIED

The source code of the smart contract was taken from the Gain repository:

<https://github.com/GainDAO/token>

Last commit: [973c76e1e8bdbeffe2444be963b9777381358090](#)

Within the scope of this audit, the team of auditors reviewed the following contract(s):

- ERC20Distribution.sol
- GainDAOToken.sol

## During the audit, Zokyo Security ensured that the contract:

- Implements and adheres to the existing standards appropriately and effectively;
- The documentation and code comments match the logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices, efficiently using resources without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the most recent vulnerabilities;
- Meets best practices in code readability, etc.

Zokyo Security has followed best practices and industry-standard techniques to verify the implementation of Gain smart contracts. To do so, the code was reviewed line by line by our smart contract developers, who documented even minor issues as they were discovered. Part of this work includes writing a test suite using the Hardhat testing framework. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

01	Due diligence in assessing the overall code quality of the codebase.	03	Testing contracts logic against common and uncommon attack vectors.
02	Cross-comparison with other, similar smart contracts by industry leaders.	04	Thorough manual review of the codebase line by line.




# Executive Summary


Contracts are well written and structured. There was no critical issue found during the audit, alongside one with medium severity, some of low severity and one informational issue . All the mentioned findings may have an effect only in case of specific conditions performed by the contract owner and the investors interacting with it. They are described in detail in the “Complete Analysis” section.

# STRUCTURE AND ORGANIZATION OF THE DOCUMENT


For the ease of navigation, the following sections are arranged from the most to the least critical ones. Issues are tagged as “Resolved” or “Unresolved” depending on whether they have been fixed or addressed. The issues that are tagged as “Verified” contain unclear or suspicious functionality that either needs explanation from the Client or remains disregarded by the Client. 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 contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.

 **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.

# COMPLETE ANALYSIS

## Findings Summary

#	Title	Risk	Status
1	Wrong Guard Function	Medium	Resolved
2	Unnecessary operation	Low	Resolved
3	Gas optimization	Low	Resolved
4	Gas optimization	Low	Resolved
5	Setter applied with no event emitted	Informational	Resolved



### Wrong Guard Function.

ERC20Distribution.sol - In the `purchaseTokens` `assert()` function was used to check the rate and balance or token. But rate and balance were calculated by parameters, so these checks can be false. `assert()` is used to check for code that should never be false, the role of `assert()` is to declare statements that you think will always hold. When an `assert()` statement fails, something very wrong and unexpected has happened, and you need to fix your code. Also, users can lose all gas when transaction fails by `assert()`, if it fails by `require`, the gas that was not used up to that moment will be reimbursed.

#### Recommendation:

Use `require()` instead of `assert()`.

### Unnecessary operation.

ERC20Distribution.sol - In the `startDistribution` there is a check to ensure that the contract balance of the distributed token and the `_total_distribution_balance` are the same. However, the next operation assigns the contract balance of the distributed token to the `_total_distribution_balance`, this makes the logic to be redundant.

#### Recommendation:

Remove this line.

```
_total_distribution_balance = _trusted_token.balanceOf(address(this));
```

**Gas optimization.**

ERC20Distribution.sol - In the `claimFiatToken`, The result of the transferring `_fiat_token` is boolean. So there is no need to compare with `true`, It will cause more gas consumption.

```
require(result==true,
  "Claim: transfer must succeed"
);
```

**Recommendation:**

```
require(result,
  "Claim: transfer must succeed"
);
```

**Gas optimization**

ERC20Distribution.sol - In the `purchaseTokens`, The result of the transferring `_fiat_token` is boolean. So there is no need to compare with `true`, It will cause more gas consumption.

```
require(
  result1 == true &&
  fiat_token.balanceOf(address(this)) - initfiatbalance == fiattoken_amount,
  "PurchaseTokens: fiat transfer must succeed"
);

require(
  result2==true &&
  inittokenbalance - trusted_token.balanceOf(address(this))==trustedtoken_amount,
  "PurchaseTokens: token transfer must succeed"
);
```

### Recommendation:

```
require(  
    result1 &&  
    fiat_token.balanceOf(address(this)) - initfiatbalance == fiattoken_amount,  
    "PurchaseTokens: fiat transfer must succeed"  
);
```

```
require(  
    result2 &&  
    inittokenbalance - trusted_token.balanceOf(address(this)) == trustedtoken_amount,  
    "PurchaseTokens: token transfer must succeed"  
);
```

INFORMATIONAL | RESOLVED

### Setter applied with no event emitted.

ERC20Distribution.sol - In the `changeKYCApprover` it is noticed that events are emitted after calling the setter methods in this contract.

### Recommendation:

Add event emitter which emit new KYC approver address.

	GainDAOToken.sol	ERC20Distribution.sol
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 Zokyo Security

As a part of our work assisting Gain in verifying the correctness of their contracts code, our team was responsible for writing integration tests using the Hardhat testing framework.

The tests were based on the functionality of the code, as well as a review of the Gain contracts requirements for details about issuance amounts and how the system handles these.

### ERC20Distribution

- ✓ Should be deployed correctly (268ms)
- ✓ Should be able to view start rate distribution
- ✓ Should be able to view end rate distribution
- ✓ Should be able to view total distribution balance
- ✓ Should be able to start distribution only when paused (189ms)
- ✓ Should be able to get current\_distributed\_balance
- ✓ Should be able to get status of distribution (69ms)
- ✓ Should be able to changr kyc approver (63ms)
- ✓ Should be able to get the fiat token balance of user
- ✓ Should be able to get the fiat token balance of contract
- ✓ Should be able to get divider rate distribution
- ✓ Should be able to claim fiat token (99ms)
- ✓ Should be able to purchase allowed (156ms)
- ✓ Should be able to get current rate (44ms)
- ✓ Should be able to purchase tokens (369ms)
- ✓ Should be able to purchase tokens- when kyc is needed (147ms)

### GainDAOToken

- ✓ Should be deployed correctly (46ms)
- ✓ Should be able to unpause (75ms)
- ✓ Should be able to mint (67ms)
- ✓ Should be able to burn (132ms)

20 passing (10s)

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	100	100	100	
GainDAOToken.sol	100	100	100	100	
All files	100	100	100	100	

We are grateful for the opportunity to work with the Gain team.

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

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

