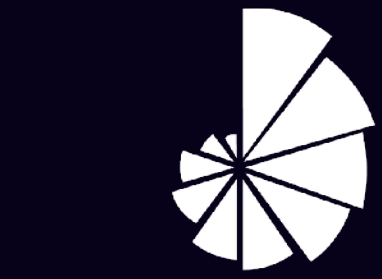

GAiN.Ai

AI POWERED POOLS

SMART CONTRACTS REVIEW

zokyo

January 22nd 2024 | v. 1.0

Security Audit Score

PASS

Zokyo Security has concluded that these smart contracts passed a security audit.

ZOKYO AUDIT SCORING GAIN

1. Severity of Issues:
    - Critical: Direct, immediate risks to funds or the integrity of the contract. Typically, these would have a very high weight.
    - High: Important issues that can compromise the contract in certain scenarios.
    - Medium: Issues that might not pose immediate threats but represent significant deviations from best practices.
    - Low: Smaller issues that might not pose security risks but are still noteworthy.
    - Informational: Generally, observations or suggestions that don't point to vulnerabilities but can be improvements or best practices.
2. Test Coverage: The percentage of the codebase that's covered by tests. High test coverage often suggests thorough testing practices and can increase the score.
3. Code Quality: This is more subjective, but contracts that follow best practices, are well-commented, and show good organization might receive higher scores.
4. Documentation: Comprehensive and clear documentation might improve the score, as it shows thoroughness.
5. Consistency: Consistency in coding patterns, naming, etc., can also factor into the score.
6. Response to Identified Issues: Some audits might consider how quickly and effectively the team responds to identified issues.

HYPOTHETICAL SCORING CALCULATION:

Let's assume each issue has a weight:
- Critical: -30 points
- High: -20 points
- Medium: -10 points
- Low: -5 points
- Informational: -1 point

Starting with a perfect score of 100:
- 0 Critical issues: 0 points deducted
- 0 High issues: 0 points deducted
- 2 Medium issues: 2 resolved = 0 points deducted
- 1 Low issue: = 1 resolved = 0 points deducted
- 6 Informational issues: 6 resolved = 0 points deducted

Hence, the score stands at 100.

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 contracts codebase for quality, security, and correctness.

Contract Status

There were 0 critical issues found during the review. (See Complete Analysis)

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

The source code of the smart contract was taken from the Gain repository:
Repo: https://github.com/GainDAO/token

Last commit -cfaaac2ef42188a61098e809d0deca55383a1336

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

- ./PaymentToken.sol
- ./ERC20Distribution.sol
- ./GainDAOToken.sol
- ./ERC20DistributionNative.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. 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.

02 Cross-comparison with other, similar smart contracts by industry leaders.

03 Thorough manual review of the codebase line by line.

Executive Summary

The contracts feature articulate and organized language. The assessment revealed no critical or high issues but did highlight instances of medium, low, and informational severity. Detailed explanations of these matters can be found in the "Comprehensive 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” or “Acknowledged” depending on whether they have been fixed or addressed. Acknowledged means that the issue was sent to the Gain team and the Gain team is aware of it, but they have chosen to not solve it. 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	Signature proof not including a domain separator	Medium	Resolved
2	No access control in startDistribution()	Medium	Resolved
3	Absence of SafeERC20 library in ERC20 asset transfer	Low	Resolved
4	Unnecessary redundant variable	Informational	Resolved
5	Increased deployment cost due to extended error messages	Informational	Resolved
6	Sender identification discrepancy	Informational	Resolved
7	claimNativeToken can be removed from ERC20Distribution.sol	Informational	Resolved
8	Inconsistency between rate calculation and comment in ERC20DistributionNative	Informational	Resolved
9	Comments for ERC20Distribution and ERC20DistributionNative are the same	Informational	Resolved

MEDIUM-1 | RESOLVED

Signature proof not including a domain separator

In ERC20DistributionNative.sol & ERC20Distribution.sol - Function `purchaseAllowed()` posses the following risk: If a contract does not use a domainSeparator when verifying signatures: bytes call data proof, it becomes susceptible to cross-domain attacks. An attacker could potentially take a valid signature from one context (chain, address, ...) and use it to perform unauthorized actions in a different context.

Recommendation:
See Definition of domainSeparator in eip-712 and follow it to construct a secure proof.

Fix: Issue is addressed in commit a356632 by adding the chain.id and contract address information to function `hashForKYC()`. According to requirements specifications of the project a full eip-712 spec is not needed.

MEDIUM-2 | RESOLVED

No access control in startDistribution()

The initial state of the ERC20Distribution.sol/ERC20DistributionNative.sol is paused (reasons are unclear, might be due to waiting for the kyc approver to be assigned first or some other factors). Keeping in mind that the contract should be paused until the devs see fit, anyone can call the startDistribution() function and unpause the system.

Recommendation:
Unpausing the contract should be done by a privileged role ideally.

LOW-1 | RESOLVED

Absence of SafeERC20 library in ERC20 asset transfers

The smart contracts under review do not incorporate the SafeERC20 library when conducting ERC20 token transfers. This has a negative impact when the ERC20 involved in the `transfer` does not return value. There exist older ERC20 that did not follow the standard `transfer` function prototype and that leads to undesirable outcomes.

Recommendation:
It is advised to use `SafeERC20` library to undergo transfers of ERC20 assets.

Fix - Issue resolved in commit `cfaaac2` .

INFORMATIONAL-1 | RESOLVED

Unnecessary redundant variable

ERC20DistributionNative - In function `purchaseTokens()` the variable `inittokenbalance` is declared in order to reflect the ERC20 balance of the contract:

```
uint256 inittokenbalance = _trusted_token.balanceOf(address(this));
```

but priorly `pool_balance` served that purpose and the values of both variables are the same on the moment they are used:

```
uint256 pool_balance = _trusted_token.balanceOf(address(this));
```

Recommendation:
Assign the quantity (i.e. ERC20 balance) once.

Increased deployment cost due to extended error messages

A significant concern has been identified during the audit, specifically related to the extensive use of detailed error messages within the smart contracts. While these detailed error messages offer valuable insights, it's essential to note that they contribute to an increase in the size of contracts. This, in turn, escalates deployment costs, which are paid in gas and increase the chance of hitting the deployment size threshold.

One example:

```
function mint(address to, uint256 amount) public {
     require(
         hasRole(MINTER_ROLE, _msgSender()),
         "GainDAOToken: _msgSender() does not have the minter role"
     );

     _mint(to, amount);
 }
```

Recommendation:
Use custom error objects (e.g. `error UnauthorizedMinter()`) or concise short error codes as revert messages. This issue is shown in all contracts within scope that use `require` statements.

Sender identification discrepancy

An issue lies in the way the `_msgSender()` function is used within the smart contracts. The `_msgSender()` function is part of the context abstract contract, and it is designed to provide a way to determine the original sender of a message in a multi-contract system.

The concern here is that using `_msgSender()` to determine the sender of the function call might not yield the intended result in the context of the contract. This inconsistency could lead to unexpected behavior.

In summary, the issue is related to inconsistent usage of `msg.sender` and `_msgSender()` across the contracts. The coding style and usage of these variables need to be aligned for clarity and correctness.

Recommendation:
To address this concern, developers should ensure that the same method is consistently used to determine the sender across all contracts. Either use `msg.sender` consistently or `_msgSender()` consistently, depending on the intended behavior.

Fix-1: The issue is partially addressed in commit `a356632`. All occurrences of _msgSender() are being replaced by msg.sender. Issue is resolved in all contracts except in GainDAOToken where both _msgSender() and msg.sender are being used together.

Fix-2:Issue fixed on commit `cfaaac2`

INFORMATIONAL-4 | RESOLVED

claimNativeToken can be removed from ERC20Distribution.sol

The ERC20Distribution contract uses a fiat token (ERC20) to purchase gain tokens, ETH cannot be received by this contract (unless forced sent vie self-destruct), therefore the function claimNativeToken is unnecessary since there can never be ETH in the contract.

Recommendation:
claimNativeToken should be removed from ERC20Distribution.sol

INFORMATIONAL-5 | RESOLVED

Inconsistency between rate calculation and comment in ERC20DistributionNative

In the currentRateUndivided function in ERC20DistributionNative contract, the _current_distributed_balance increases after each call to purchaseTokens, so the offset_e18 (which is _total_distribution_balance - _current_distributed_balance) decreases. This calculation causes the current rate to decrease as well, assuming that all other variables remain constant during the single distribution. However, the comment suggests that the rate should be ascending, i.e., it should increase as _current_distributed_balance increases. In this implementation, the opposite happens: the rate decreases as _current_distributed_balance increases, which is contrary to the intended "ascending fractional linear rate" behavior.

Recommendation:
Use the adjusted formula or change the mentioned comment.
Client comment: This issue is by design.

Comments For ERC20Distribution and ERC20DistributionNative are the same

There are instances in the ERC20Distribution contract where the comments correspond to the ERC20DistributionNative, such as comments L272 and L273, they mention ether while it should be the fiat token.

Recommendation:
The comments should be corrected.

	./PaymentToken.sol ./ERC20Distribution.sol ./GainDAOToken.sol ./ERC20DistributionNative.sol
Reentrance	Pass
Access Management Hierarchy	Pass
Arithmetic Over/Under Flows	Pass
Unexpected Ether	Pass
Delegate Call	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

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.

zokyo