COLEND

SMART CONTRACTS REVIEW

zokyo

January 7th 2025 | v. 1.0

Security Audit Score

PASS

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

ZOKYO AUDIT SCORING COLEND

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.

SCORING CALCULATION:

Let's assume each issue has a weight:
- Critical: -30 points
- High: -20 points
- Medium: -10 points
- Low: -5 points
- Informational: 0 points

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

Thus, 100 - 2 = 98

TECHNICAL SUMMARY

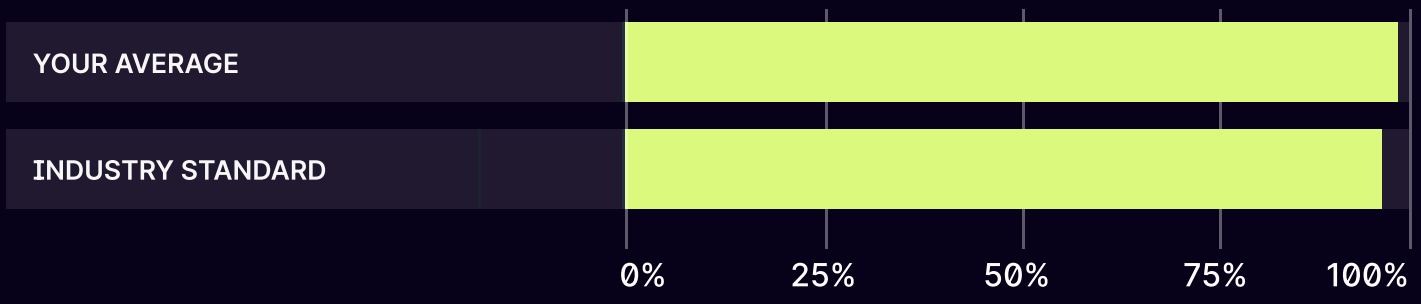
This document outlines the overall security of the Colend smart contract/s evaluated by the Zokyo Security team.

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

Contract Status

There were 0 critical issues found during the review. (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 contract/s 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 Colend team put in place a bug bounty program to encourage further active analysis of the smart contract/s.

Table of Contents

AUDITING STRATEGY AND TECHNIQUES APPLIED

The source code of the smart contract was taken from the Colend repository:
Repo: https://github.com/Colend-Protocol/token-geyser

Last commit - e1542e6cdbb037e857165c2a4dd682bb37331d17

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

- Geyser.sol
- COLEND.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 Colend smart contract/s. 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 contract/s logic against common and uncommon attack vectors.
02	Cross-comparison with other, similar smart contract/s by industry leaders.	04	Thorough manual review of the codebase line by line.

Executive Summary

The smart contracts implement a staking system where users can deposit tokens into specific vaults to earn rewards over time. The system is designed to manage multiple vaults, each with its own set of staking parameters. Core functionalities include adding new stakes, tracking the total stake and stake units, and allowing users to exit the system via functions like `rageQuit`. The contracts leverage events for transparency and include mechanisms to calculate and update staking-related metrics efficiently. The design aims to ensure scalability and precise tracking of user interactions within the staking ecosystem.

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 Colend team and the Colend 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	Missing call to _disableInitializers	Medium	Resolved
2	Lack of Two-Step Ownership Transfer	Low	Acknowledged
3	The owner can renounce ownership	Low	Acknowledged
4	Unnecessary Use of SafeMath	Informational	Resolved
5	Unused error declaration	Informational	Acknowledged

MEDIUM-1 | RESOLVED

Missing call to _disableInitializers

An uninitialized contract can be taken over by an attacker. This applies to both a proxy and its implementation contract, which may impact the proxy. To prevent the implementation contract from being used, you should invoke the `_disableInitializers()` function in the constructor to automatically lock it when it is deployed, more information can be found here.

Recommendation:

Consider calling `_disableInitializers()` in the constructor on Geyser.sol.

LOW-1 | ACKNOWLEDGED

Lack of Two-Step Ownership Transfer

The Geyser contract does not implement a two-step process for transferring ownership. In its current state, ownership can be transferred in a single step, which can be risky as it could lead to accidental or malicious transfers of ownership without proper verification.

Recommendation:

Implement a two-step process for ownership transfer where the new owner must explicitly accept the ownership. It is advisable to use OpenZeppelin's Ownable2StepUpgradeable.

Client comment: This is optional since every `transfer ownership` transaction will be done by multisig / DAO wallet. But if you still want us to move to Ownable2StepUpgradeable for better secure flow, it's fine to do that.

LOW-2 | ACKNOWLEDGED

The owner can renounce ownership

The `Ownable` contracts includes a function named `renounceOwnership()` which can be used to remove the ownership of the contract.

If this function is called on the `Geyser` contract, it will result in the contract becoming disowned. This would subsequently break functions of the token that rely on `onlyOwner` modifier.

Recommendation:
override the function to disable its functionality, ensuring the contract cannot be disowned e.g.

Client comment: This is optional, we can keep `renounch ownership` since all transactions will be done by multisig / DAO.

INFORMATIONAL-1 | RESOLVED

Unnecessary Use of SafeMath

The contract imports `SafeMath` from the OpenZeppelin library, but in Solidity `0.8.0` and later, arithmetic operations are checked for overflow and underflow by default. Using `SafeMath` is redundant and not necessary, as the compiler automatically handles these checks.

Recommendation:
Remove the `SafeMath` import and refactor the contract to remove any reliance on it, as Solidity `0.8+` provides built-in overflow and underflow checks.

Unused error declaration

The COLEND contract declares ZeroAddress error but it's not used in the contract.

Recommendation:
Remove an unused error in the contract.

Client comment: This is optional, we can keep it or remove it in next upgradeable version

	Geyser.sol COLEND.sol
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 Zokyo Security

As a part of our work assisting Colend Protocol 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 Colend Protocol contracts requirements for details about issuance amounts and how the system handles these.

ERC1271

isValidSignature

✓ should return error value if signed by account other than owner
✓ should revert if signature has incorrect length
✓ should return success value if signed by owner

Geyser

initialize

initializer modifier

✓ should prevent multiple initializations (64ms)

when rewardScaling.floor > rewardScaling.ceiling

✓ should fail (44ms)

when rewardScalingTime = 0

✓ should fail (46ms)

when parameters are valid

✓ should set contract variables (55ms)

initializeLock

✓ should allow initialization once
✓ should prevent multiple initializations

admin functions

fundGeyser

with insufficient approval

✓ should fail

with duration of zero

✓ should fail

as user

✓ should fail

when offline

✓ should fail

when shutdown

✓ should fail

when online
at first funding
✓ should succeed
✓ should update state correctly
✓ should emit event
✓ should transfer tokens
at second funding
with no rebase
✓ should succeed
✓ should update state correctly
✓ should emit event
✓ should transfer tokens
after unstake
with partial rewards exausted
✓ should succeed
✓ should update state correctly
✓ should emit event
✓ should transfer tokens
with full rewards exausted
✓ should succeed
✓ should update state correctly
✓ should emit event
✓ should transfer tokens
isValidVault
when no factory registered
✓ should be false
when vault from factory registered
✓ should be true
when vault from factory removed
✓ should be false
when vault not from factory registered
✓ should be false
when vaults from multiple factory registered
✓ should be true
registerVaultFactory
as user
✓ should fail
when online
✓ should succeed
✓ should update state
✓ should emit event

when offline
✓ should succeed
✓ should update state
✓ should emit event
when shutdown
✓ should fail
when already added
✓ should fail
when removed
✓ should succeed
✓ should update state
✓ should emit event
with second factory
✓ should succeed
✓ should update state
✓ should emit event
removeVaultFactory
as user
✓ should fail
when online
✓ should succeed
✓ should update state
✓ should emit event
when offline
✓ should succeed
✓ should update state
✓ should emit event
when shutdown
✓ should fail
when never added
✓ should fail
when already removed
✓ should fail
registerBonusToken
as user
✓ should fail
when online
on first call
with address zero
✓ should fail
with geyser address
✓ should fail

with geyser address
✓ should fail
with staking token
✓ should fail
with reward token
✓ should fail
with rewardPool address
✓ should fail
with bonus token
✓ should succeed
✓ should update state
✓ should emit event
on second call
with same token
✓ should fail
with different bonus token
✓ should succeed
✓ should update state
✓ should emit event
when offline
✓ should fail
when shutdown
✓ should fail
when registering maximum number of bonus tokens
✓ should succeed for MAX_REWARD_TOKENS (92ms)
✓ should fail when exceeding MAX_REWARD_TOKENS (84ms)
rescueTokensFromRewardPool
as user
✓ should fail
with reward token
✓ should fail
with bonus token
✓ should fail
with staking token
✓ should succeed
✓ should transfer tokens
with geyser as recipient
✓ should fail
with staking token as recipient
✓ should fail
with reward token as recipient
✓ should fail

with rewardPool as recipient
✓ should fail
with address 0 as recipient
✓ should fail
with other address as recipient
✓ should succeed
✓ should transfer tokens
with zero amount
✓ should succeed
✓ should transfer tokens
with partial amount
✓ should succeed
✓ should transfer tokens
with full amount
✓ should succeed
✓ should transfer tokens
with excess amount
✓ should fail
when online
✓ should succeed
✓ should transfer tokens
when offline
✓ should fail
when shutdown
✓ should fail
user functions
stake
when offline
✓ should fail
when shutdown
✓ should fail
to invalid vault
✓ should fail
with amount of zero
✓ should fail
with insufficient balance
✓ should fail
when not funded
✓ should succeed
when funded
on first stake
as vault owner

✓ should succeed
✓ should update state
✓ should emit event
✓ should lock tokens
on second stake
✓ should succeed
✓ should update state
✓ should emit event
✓ should lock tokens
when MAX_STAKES_PER_VAULT reached
✓ should fail
when stakes reset
✓ should succeed
✓ should update state
✓ should emit event
✓ should lock tokens
unstake
with default config
when offline
✓ should fail
when shutdown
✓ should fail
with invalid vault
✓ should succeed
with permissioned not signed by owner
✓ should fail
with amount of zero
✓ should fail
with amount greater than stakes
✓ should fail
with fully vested stake
✓ should succeed
✓ should update state
✓ should emit event
✓ should transfer tokens
✓ should unlock tokens
with partially vested stake
✓ should succeed
✓ should update state
✓ should emit event
✓ should transfer tokens
✓ should unlock tokens

with floor and ceiling scaled up

✓ should succeed
✓ should update state
✓ should emit event
✓ should transfer tokens
✓ should unlock tokens

with no reward

✓ should succeed
✓ should update state
✓ should emit event
✓ should unlock tokens

with partially vested reward

✓ should succeed
✓ should update state
✓ should emit event
✓ should transfer tokens
✓ should unlock tokens

with flash stake

✓ should succeed
✓ should update state
✓ should emit event
✓ should lock tokens
✓ should unlock tokens

with one second stake

✓ should succeed
✓ should update state
✓ should emit event
✓ should transfer tokens
✓ should unlock tokens

with partial amount from single stake

✓ should succeed
✓ should update state
✓ should emit event
✓ should transfer tokens
✓ should unlock tokens

with partial amount from multiple stakes

✓ should succeed
✓ should update state
✓ should emit event
✓ should transfer tokens
✓ should transfer tokens

with full amount of the last of multiple stakes

✓ should succeed
✓ should update state
✓ should emit event
✓ should transfer tokens
✓ should unlock tokens

with full amount of multiple stakes

✓ should succeed
✓ should update state
✓ should emit event
✓ should transfer tokens
✓ should unlock tokens

when one bonus token

with no bonus token balance

✓ should succeed
✓ should update state
✓ should emit event
✓ should transfer tokens
✓ should unlock tokens

with fully vested stake

✓ should succeed
✓ should update state
✓ should emit event
✓ should transfer tokens
✓ should unlock tokens

with partially vested stake1

✓ should succeed
✓ should update state1
✓ should emit event
✓ should transfer tokens
✓ should unlock tokens

with multiple vaults

✓ should succeed (50ms)
✓ should update state (45ms)
✓ should emit event (79ms)
✓ should transfer tokens (66ms)
✓ should unlock tokens (62ms)

rageQuit

when online

✓ should succeed
✓ should update state
✓ should transfer tokens

when offline
✓ should succeed
✓ should update state
when shutdown
✓ should succeed
✓ should update state
with unknown vault
✓ should fail
when no stake
✓ should fail
when insufficient gas
✓ should fail
when insufficient gas with multiple stakes
✓ should fail
when single stake
✓ should succeed
✓ should update state
when multiple stakes
✓ should succeed
✓ should update state
Geyser Reward and Stake Unit Calculations
✓ should correctly calculate current unlocked rewards
✓ should correctly calculate future unlocked rewards
✓ should correctly calculate current vault reward
✓ should correctly calculate future vault reward
✓ should correctly calculate current stake reward
✓ should correctly calculate future stake reward
✓ should correctly calculate future vault stake units

PowerSwitch
powerOn
✓ should fail if msg.sender is not admin
✓ should succeed if in offline state
✓ should fail if in online state
✓ should fail if in shutdown state
✓ should succeed and emit event
powerOff
✓ should fail if msg.sender is not admin
✓ should succeed if in offline state
✓ should fail if in online state
✓ should fail if in shutdown state
✓ should succeed and emit event

emergencyShutdown
- ✓ should fail if msg.sender is not admin
- ✓ should succeed if in offline state
- ✓ should fail if in online state
- ✓ should fail if in shutdown state
- ✓ should succeed and emit event

Powered

getPowerSwitch
- ✓ should succeed

getPowerController
- ✓ should succeed

onlyOnline
- ✓ hould succeed if online
- ✓ should fail if offline
- ✓ should fail if shutdown

onlyOffline
- ✓ should fail if online
- ✓ should succeed if offline
- ✓ should fail if shutdown

notShutdown
- ✓ should succeed if online
- ✓ should succeed if offline
- ✓ should fail if shutdown

onlyShutdown
- ✓ should fail if online
- ✓ should fail if offline
- ✓ hould succeed if shutdown

RewardPool

sendERC20
- ✓ should succeed if msg.sender is admin
- ✓ should fail if msg.sender is controller
- ✓ should succeed if online
- ✓ should fail if offline
- ✓ should fail if shutdown
- ✓ should succeed with full balance
- ✓ should succeed with partial balance
- ✓ should succeed with no balance

rescueERC20
- ✓ should fail if msg.sender is admin
- ✓ should succeed if msg.sender is controller
- ✓ should fail if online
- ✓ should fail if offline

✓ should succeed if shutdown
✓ should fail if recipient is not defined
✓ should succeed with single token
✓ should succeed with 100 tokens (1735ms)

COLEND

✓ should deploy with correct initial values (39ms)
✓ should transfer tokens
✓ should permit
✓ should upgrade (107ms)
✓ should revert when recipient is zero address
✓ should deploy successfully with non-zero address
✓ should revert when trying to initialize twice (38ms)
✓ should burn tokens correctly (64ms)
✓ should only allow UPGRADER_ROLE to upgrade (76ms)

UniversalVault

nft

✓ should succeed

getNonce

✓ should succeed

owner

✓ should succeed

getLockSetCount

✓ should succeed

getLockAt

✓ should fail when no locks

getBalanceDelegated

✓ should succeed

getBalanceLocked

✓ should succeed

checkBalances

✓ should succeed

lock

with incorrect permission

✓ should fail when wrong signer
✓ should fail when wrong function signature
✓ should fail when wrong delegate
✓ should fail when wrong token
✓ should fail when wrong amount
✓ should fail when wrong nonce

with correct permission

✓ should succeed
✓ should create lock if new delegate-token pair
✓ should update lock if existing delegate-token pair

✓ should fail if insufficient vault balance on new lock
✓ should fail if insufficient vault balance on existing lock
✓ should bump nonce
✓ should emit event

when owner is ERC1271 compatible smart contract

with valid wallet

✓ should succeed
✓ should emit event

with invalid wallet

✓ should fail

unlock

with incorrect permission

✓ should fail when wrong signer
✓ should fail when wrong function signature
✓ should fail when wrong delegate
✓ should fail when wrong token
✓ should fail when wrong amount
✓ should fail when wrong nonce

with correct permission

✓ should succeed
✓ should fail if lock does not exist
✓ should update lock balance if amount < balance
✓ should delete lock if amount >= balance
✓ should bump nonce
✓ should emit event

when owner is ERC1271 compatible smart contract

with valid wallet

✓ should succeed
✓ should emit event

with invalid signature

✓ should fail

rageQuit

as non-owner

✓ should fail

with insufficient gas forwarded

✓ should fail

delegate with success

✓ should succeed
✓ should fail when lock does not exist
✓ should delete lock data
✓ should emit event
✓ should return data

```
    delegate with revert
      ✓ should succeed
      ✓ should delete lock data
      ✓ should emit event
      ✓ should return data
    delegate with revert message
      ✓ should succeed
      ✓ should delete lock data
      ✓ should emit event
      ✓ should return data
    delegate with out of gas error
      ✓ should succeed (229ms)
      ✓ should delete lock data (257ms)
      ✓ should emit event (152ms)
      ✓ should return data
    delegate is EOA
      ✓ should succeed
      ✓ should delete lock data
      ✓ should emit event
      ✓ should return data
ERC20
    ERC20:transfer
      ✓ should succeed
      ✓ should transfer tokens
      ✓ should fail if insufficient unlocked balance
  ETH
    ETH:receive
      ✓ should succeed
      ✓ should receive correct amount
    ETH:send
      ✓ should succeed
      ✓ should send correct amount
      ✓ should fail if insufficient amount
VaultFactory
  getTemplate
      ✓ should succeed
  create
      ✓ should succeed
      ✓ should successfully call owner
  create2
      ✓ should succeed
      ✓ should successfully call owner


    349 passing (1m)
```

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

FILE	% STMTS	% BRANCH	% FUNCS	% LINES
COLEND.sol	100	100	100	100
Geyser.sol	100	98.84	100	100
All Files	**100**	**98.91**	**100**	**100**

We are grateful for the opportunity to work with the Colend 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 Colend team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

zokyo