SMART CONTRACT AUDIT

ZOKYO.

January 26th, 2022 | 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

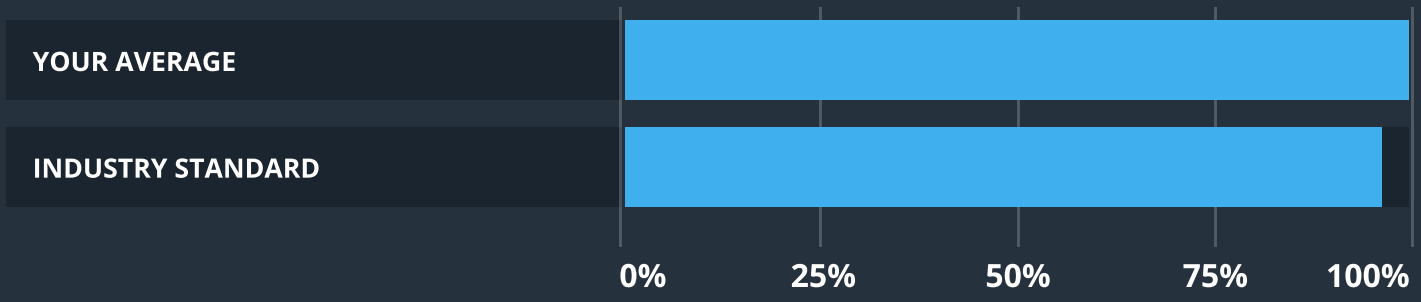
This document outlines the overall security of the Goldefy smart contracts, evaluated by Zokyo's Blockchain Security team.

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

Contract Status

There were critical, high and medium issues found during the audit.

Testable Code



The testable code is 99%, 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 security of the contract we at Zokyo recommend that the Goldefy 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 Goldefy smart contract's source code was taken from the smart contract provided by the Goldefy team

SHA-256 (audited):
StakingB.sol F2BEEBAE21E55066C32B8BF610493258257F8E647978F5E043A600E3FCE5BAB5
StakingB_1.sol: A4350160BCFD1537FBC652E310EA7372110E5AC64A9C10A7FA882F178ECB1722

SHA-256 (post-audited):
FixedStaking.sol FC0EC339C3B3CECDDA1D481FE6D879252B850D4899FBC5FADA556263492F0E31
FlexibleStaking.sol: 9738E94EE0CD88530757D4C6183638D5F2BD7B0074C27F192B9AD9A227873B86

Within the scope of this audit Zokyo auditors have reviewed the following contract(s):

- FlexibleStaking.sol (prev. StakingB_1.sol)
- FixedStaking.sol (prev. StakinB.sol)

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

- Implements and adheres to existing standards appropriately and effectively;
- Documentation and code comments match logic and behavior;
- Follows best practices in efficient use of resources, 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 Goldefy 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. 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.

EXECUTIVE SUMMARY

There were critical, high and medium issues found during the audit, they relate to the following:

- Logic issues
- Risk of underflow
- Risk of overflow
- View function will never return value.

After recommendations by Zokyo auditors and discussion with the Customer it was decided that all these issues don’t influence security and efficiency of the smart contract provided by Goldefy Team.

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 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

CRITICAL | RESOLVED

Logic issue.

StakingB_1.sol Line 512
“lastRewardBlock” must be initialized when the pool is set. In other case first reward will be counted as a reward in range [0; last block number] that is a numerous amount.

Recommendation:
Initialize lastRewardBlock in pool setting.

CRITICAL | RESOLVED

Logic issue.

StakingB_1.sol Line 614-615
First “updatePool” function executes “pool.lastRewardBlock = block.number;” and than “pool.accTokenPerShare += (((block.number - pool.lastRewardBlock) * pool.rate...” where in case of first operation “block.number” becomes equal to “pool.lastRewardBlock” their sunstraction in next operation will always cause zero, that is the coefficient for the whole equation. Pending rewards will never be increased.

Recommendation:
Place the first operation after the second.

CRITICAL | RESOLVED

Risk of underflow.

StakingB_1.sol Line 579
Each user can use the unstake function even if they have nothing to unstake, so everyone can decrease the “amountOfUsers” variable that can cause an underflow and won’t allow other users to unstake.

Recommendation:
Check if “userInfo[msg.sender]” has not been already deleted before decreasing an “amountOfUsers”.

HIGH | RESOLVED

Risk of overflow.

accTokenPerShare calculation utilizes 1e12*1e18*1e32 that is equal to 1e62. Max size of unsigned integer (256) is 2**256-1 that is ~1e77. With high pool rate and low rateDenominator such amount can cause an overflow.

Recommendation:
Handle or prevent an overflow exception.

MEDIUM | RESOLVED

View function will never return value.

StakingB.sol Line 46
“stakesInfo” will never return value because it creates an array [from; to) but tries to iterate through [from; to].

Recommendation:
Change the format of array to
“ s = new Stake[](_to - _from+1)” or
“for (uint256 i = _from; i < _to; i++)”

LOW | RESOLVED

Excess functionality.

StakingB_1.sol Line 632
Calling pool.token will cause the same result as the tokenAddress() function.

Recommendation:
Delete tokenAddress() function.

INFORMATIONAL | RESOLVED

Improper use of access modifiers.

The whole functions in contract use “public” access modifiers but most of them can be called only externally.

Recommendation:
Use “external” access modifier in functions that can be called only externally.

INFORMATIONAL | RESOLVED

The Solidity version should be updated.

Best practices for Solidity development and auditors standard checklist requires strict and explicit usage of the latest stable version of Solidity, which is 0.8.11 at the moment.

Recommendation:
Consider updating to “pragma solidity 0.8.11;”.

INFORMATIONAL | RESOLVED

Code structure is messed up.

StakingB_1.sol Line 524
Storage and events must be separated in order to make code structure more readable.

Recommendation:
Move the “amountOfUser” variable to the other variables.

INFORMATIONAL | RESOLVED

No message in “require” statements.

StakingB_1.sol Line 546
StakingB.sol Lines 80, 95, 97, 99, 102, 103, 115, 117
To hold the exception error messages must be added.

Recommendation:
Add an error message to each exception.

INFORMATIONAL | RESOLVED

Code could be simplified.

StakingB_1.sol Line 609
“pool.lastRewardBlock = block.number” can be executed before condition so this operation shouldn’t be repeated twice in the code.

Recommendation:
Use just one realisation of “pool.lastRewardBlock = block.number” before condition.

INFORMATIONAL | RESOLVED

Use constants.

StakingB.sol Lines 17, 18, 19
StakingB_1.sol Lines 518
If the value of the storage variable can’t be changed it can be set as constant in order to save gas.

Recommendation:
Set offered storage as constant

	FlexibleStaking.sol	FixedStaking.sol
Re-entrancy	Pass	Pass
Access Management Hierarchy	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass
Delegatecall Unexpected Ether	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 Secured team

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

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

Contract: **FlexibleStaking**
- Staking
  - ✓ stakes token (780ms)
  - ✓ doesn't stake 0 token (658ms)
  - ✓ doesn't allow to stake twice from the same address (643ms)
- Unstaking
  - ✓ unstakes tokens (784ms)
  - ✓ only stakers can unstake tokens (602ms)
  - ✓ doesn’t unstakes tokens when reserves are empty (623ms)
  - ✓ force unstake (680ms)
  - ✓ doesn't allow to force unstake if contract has enough tokens to unstake in normal way (608ms)
  - ✓ sets new fee address (144ms)
  - ✓ shows pending reward for a user right after staking (602ms)
  - ✓ show pending reward for a user (825ms)
  - ✓ doesn't allow to set rate greater than 1e6 (92ms)

Contract: **FixedStaking**
- Staking
  - ✓ stakes token (1175ms)
  - ✓ doesn't stake with invalid stake class (538ms)
  - ✓ doesn't stake more stakes than MAX_STAKES (1131ms)
  - ✓ doesn't stake when not enough reward left (280ms)
  - ✓ doesn't stake when tx is failed (448ms)
- Unstaking
  - ✓ unstakes tokens (789ms)
  - ✓ doesn't unstake the same stake twice (794ms)
  - ✓ doesn't unstake if staking period hasn't ended (635ms)
- View methods
  - ✓ shows user stakes (883ms)
  - ✓ shows number of all stakes (1514ms)
  - ✓ shows information about all stakes (1614ms)

```
✓ shows information about range of stakes (1726ms)
✓ shows correct count of active stakes when some of stakes where unstaked (1826ms)
✓ transfers ownership only from owner (1064ms)
✓ returns accidentally Sent tokens (805ms)
✓ updates max number of staking per user (212ms)
✓ changes fee address (396ms)

29 passing (28s)
```

FILE	% STMTS	% BRANCH	% FUNCS
FlexibleStaking.sol	98.11	95	100.00 (Line 611 - unreachable)
FixedStaking.sol	100.00	100.00	100.00
All files	**99.05**	**97,5**	**100**

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

ZOKYO.