SMART CONTRACT AUDIT

ZOKYO.

August 25th, 2021 | v. 2.0

PASS

Zokyo's Security Team has concluded that this smart contract passes security qualifications to be listed on digital asset exchanges.

TECHNICAL SUMMARY

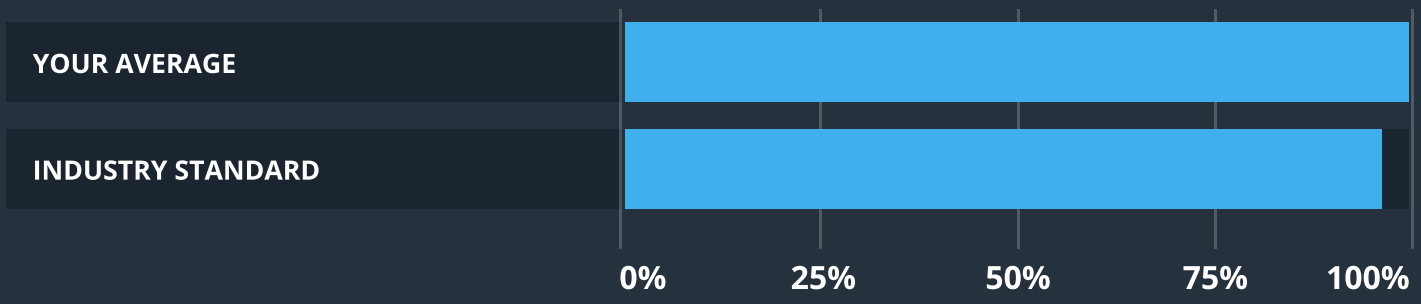
This document outlines the overall security of the FNDZ smart contracts, evaluated by Zokyo's Blockchain Security team.

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

Contract Status

There were no critical or high 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 security of the contract we at Zokyo recommend that the FNDZ 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 FNDZ smart contract's source code was taken from one archive provided by the FNDZ team:
SHA-256 audited:
657657a33dea83dab1ccf2b72194841c2447ed67a46d37b163b9f9a2d588be41
SHA-256 post-audit:
a09ba15abdc9ec2bca99879f9ded14a7f9313bf15100b303ffed1bc1d8dd19d9

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

- FNDZToken.sol
- Vesting.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 FNDZ 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.
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.

EXECUTIVE SUMMARY

There were no critical or high issues found during the audit. Although certain number of medium and low issues were discovered, they relate to the following:

- Update of Solidity version
- Low level reentrancy
- Interfaces instead of contracts
- Zero address checking
- Extra method omission
- Outdated codebase.

After recommendations by Zokyo auditors, all issues that influence security and efficiency were fixed by FNDZ 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

MEDIUM | RESOLVED

Update Solidity version

The project utilises Solidity 0.5.17 and 0.4.23 versions which are marked as outdated. The standard auditor's checklist states for the usage of the latest version of the Solidity - 0.6.12, 0.7.6 or 0.8.6. Thus the issue is marked as Medium.

Recommendation:
Use the latest stable release of Solidity. Also, use the single Solidity version within the contracts set.

LOW | RESOLVED

Low level reentrancy

Vesting.sol, line 559, removeTokenGrant()
External call to the token is performed before the storage clean up. The best practices is to prepare all storage operations before the transfer is performed. Thus, in case of further development and change of the Token contract, the Vesting contract can be drained.

Recommendation:
Move all storage operations before the token transfer.

LOW | RESOLVED

Interfaces instead of contracts

Vesting.sol: DSAuthority, DSAuthEvents, ERC20Events, ERC20, ERC20Extended are defined as contracts. Though they do not have functionality, so it is recommended to define them as interfaces. Also, for ERC20 functionality, it is recommended to use OpenZeppelin.

Recommendation:
Use the latest stable release of Solidity. Also, use the single Solidity version within the contracts set.

LOW | RESOLVED

Zero address check

contracts/FNDZToken.sol#355, constructor()
Check for zero address is missing. Since there is no way to retrieve tokens from the zero address consider additional check. Thus the issue is marked as low.
The same issue refers to Vesting contract (Vesting.sol, line 465) - recipient is not checked against zero address.

Recommendation:
Add check for the zero address.

LOW | RESOLVED

Outdated codebase, consider OZ approach

Contract FNDZToken.sol is compatible with ERC20 (BEP20) standard with no additional functionality except tokens initial mint. Consider usage of standard OpenZeppelin token implementation. Inherit standard functionality with only constructor changed - such an approach increases the code readability and overall quality and decreases the size of the codebase.
The same functionality (especially for mint and burn functionality) should be considered to implement for the Token and DSTokenBase contracts from Vesting.sol.
Also consider usage of standard Ownable (for Vesting and Token contracts) and Pausable (for the token contract) functionality.

Recommendation:
Consider usage of the standard contract to omit possible mistakes in further development and to increase the code quality.

INFORMATIONAL | RESOLVED

Gas optimization

contracts/Vesting.sol#556, removeTokenGrant()
daysVested variable may be omitted, since it is not used.

Recommendation:
Remove extra variable.

INFORMATIONAL | RESOLVED

Incorrect info stored

contracts/Vesting.sol#610, calculateGrantClaim()
In case if the user has claimed tokens at least once before the vesting end, the incorrect number will be stored in case if the last claim will be performed after the vesting end. The function calculateGranClaim() returns the number of days of vesting without the consideration of tokenGrant.daysClaimed. Though, since this edge case does not influence the main flow, but only the stored info after the vesting end, the issue is marked as info.

Recommendation:
correct the calculation for the edge case.

	FNDZToken.sol	Vesting.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 FNDZ 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 FNDZ contract requirements for details about issuance amounts and how the system handles these.

```
Contract: Vesting
    ✓ Should revert with unsuccessful vested amount transfer in claimVestedTokens (189ms)
  Remove token grant test
    ✓ should revert with unsuccessful recipient transfer (69ms)
    ✓ should revert with unsuccessful refund recipient transfer (62ms)
  Claim vested tokens test
    ✓ should claim correct amount of tokens for the vested grant (2 days elapsed)
    ✓ should claim correct amount of tokens for the vested grant (3 days elapsed)
    ✓ should claim correct amount of tokens for the vested grant (5 days elapsed)
    ✓ should claim correct amount of tokens for the vested grant (7 days elapsed)
    ✓ should claim correct amount of tokens for the vested grant (10 days elapsed)
Contract: FNDZToken
    ✓ Mintable function should return false
    ✓ Should revert transferFrom when sender is zero address
    ✓ Should revert transferFrom when recipient is zero address
    ✓ Should revert approve when owner is zero address
12 passing
```

FILE	% STMTS	% BRANCH	% FUNCS	% LINES	UNCOVERED LINES
FNDZToken.sol	100.00	100.00	100.00	100.00	
Vesting.sol	100.00	100.00	100.00	100.00	
All files	**100**	**100**	**100**	**100**	

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

ZOKYO.