Domination Finance

SMART CONTRACT AUDIT

ZOKYO.

Sep 3rd, 2021 | 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

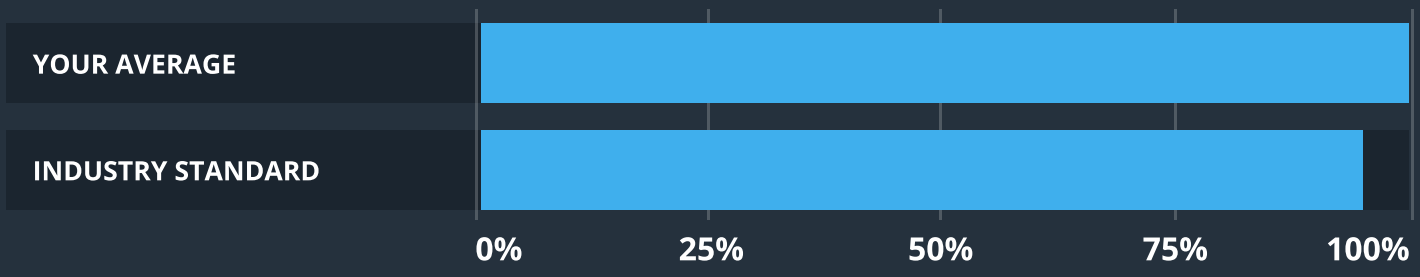
This document outlines the overall security of the DomFi smart contracts, evaluated by Zokyo's Blockchain Security team.

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

Contract Status

There were no critical 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 secure contract that's able to withstand the Ethereum network's fast-paced and rapidly changing environment, we at Zokyo recommend that the DomFi 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 Smart contract's source code was taken from the DomFi repository.

Repository (archive hash):
5038bf3c0aeeced27ecfd4a8738f4d8bb7ee8747

Last commit (archive hash):
c52a27acb7b33293b6e6834c2f4c69eaf9fd82b4

Contracts:

- Staking.sol (Constants.sol, Errors.sol, Modifiers.sol)
- DominationToken.sol
- Vester.sol

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

- Implements and adheres to existing Token standards appropriately and effectively;
- Documentation and code comments match logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices in efficient use of gas, 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 DomFi 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 using the Truffle testing framework. 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.

SUMMARY

There were neither critical issues nor issues with the high severity found during the audit. All the mentioned findings may have an effect only in case of specific conditions. They are described in detail in the “Complete Analysis” section.

The contracts are in excellent condition. They are well written and structured.
All the issues we found were successfully resolved by DomFi team. Hence, the findings bear no impact on contract performance or security, so the contracts are fully production-ready.

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 ability of the contract to compile or operate in a significant way.

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

Anyone can initialize contract at Staking.sol

MEDIUM | RESOLVED

Function initialize() can be called by anyone and if it fit all requires can set unexpected staking start timestamp.

Recommendation:
Use modifier onlyOwner to restrict function use to the owner.

Pragma is not locked to specific version

LOW | RESOLVED

Since not all the EVM compiler versions support all the features, especially the latest one's which are kind of beta versions, So the intended behavior written in code might not be executed as expected. Locking the pragma helps ensure that contracts do not accidentally get deployed using, for example, the latest compiler which may have higher risks of undiscovered bugs.

Recommendation:
Replace pragma solidity ^0.8.0 with pragma solidity 0.8.5 at FixedPoint.sol, DominationToken.sol, Vester.sol (or you can change solidity version for all contracts to 0.8.4 to fix issue with incorrect warnings during analysis "Unreachable code").

Improper Handling of ERC20 Transfers at Vester.sol and Staking.sol

LOW | RESOLVED

The linked statements invoke the transfer method without validating the expected return bool variable.

Vester.sol:

```
AccessControl(DomToken).grantRole(TRANSFER_ROLE, address(vester));
IERC20(DomToken).transfer(address(vester), vestingAmount);
```

```
function claim() public {
        require(block.timestamp >= vestingCliff, 'Vester::claim: not time yet');
        require(block.timestamp >= lastUpdate + timeout || lastUpdate == vestingBegin, 'Vester::claim:
cooldown');
        uint amount;
        if (block.timestamp >= vestingEnd) {
            amount = dom.balanceOf(address(this));
        } else {
            amount = vestingAmount * (block.timestamp - lastUpdate) / (vestingEnd - vestingBegin);
            lastUpdate = block.timestamp;
        }
        dom.transfer(recipient, amount);
}
```

Vester.sol:

```
if (partialRewards > 0) {
        DOM_TOKEN.transfer(user, partialRewards);
}
```

Recommendation:
Use a wrapper library such as SafeERC20.sol by OpenZeppelin to opportunistically evaluate the returned bool of EIP-20 transfer invocations.

Gas optimization

LOW | RESOLVED

In constructor at Staking.sol you use several requires which in final result must meet one condition.

Recommendation:
You can optimize code as shown below:

```
if (owner != _msgSender()) {
        transferOwnership(owner);
        }
require(totalDOM > 0, ERROR_ZERO_AMOUNT);
require(stakingStart > block.timestamp, ERROR_PAST_TIMESTAMP);
require(
        lspExpiration - STAKING_START_TIMESTAMP > REWARD_PERIOD,
        ERROR_EXPIRES_TOO_SOON
);
TOTAL_DOM = totalDOM;
STAKING_START_TIMESTAMP = stakingStart;
LSP_EXPIRATION = lspExpiration;
LP_TOKEN = IERC20(lpToken);
DOM_TOKEN = IERC20(domToken);
```

Additional check is required at Staking.sol

LOW | RESOLVED

In function stakeFor() there is no verification for the zero address for the beneficiary address.

Recommendation:
Add check for the zero address for beneficiary to internal function _stakeFor().

SPDX license identifier not provided in source file Vester.sol

INFORMATIONAL | RESOLVED

Recommendation:
Add SPDX license identifier.

	Staking	Vester	DominationToken
Re-entrancy	Pass	Pass	Pass
Access Management Hierarchy	Pass	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass	Pass
Unexpected Ether	Pass	Pass	Pass
Delegatecall	Pass	Pass	Pass
Default Public Visibility	Pass	Pass	Pass
Hidden Malicious Code	Pass	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass	Pass
External Contract Referencing	Pass	Pass	Pass
Short Address/ Parameter Attack	Pass	Pass	Pass
Unchecked CALL Return Values	Pass	Pass	Pass
Race Conditions / Front Running	Pass	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass	Pass
Floating Points and Precision	Pass	Pass	Pass
Tx.Origin Authentication	Pass	Pass	Pass
Signatures Replay	Pass	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass	Pass

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Zokyo Security team

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

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

Code Coverage

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

FILE	% STMTS	% BRANCH	% FUNCS	% LINES	UNCOVERED LINES
contracts\	100.00	95.45	100.00	100.00	
DominatonToken.sol	100.00	100.00	100.00	100.00	
Vester.sol	100.00	94.44	100.00	100.00	
contracts\staking\core	100.00	100.00	100.00	100.00	
Staking.sol	100.00	100.00	100.00	100.00	
contracts\staking\utils	100.00	100.00	100.00	100.00	
Constants.sol	100.00	100.00	100.00	100.00	
Errors.sol	100.00	100.00	100.00	100.00	
Modifiers.sol	100.00	100.00	100.00	100.00	
All files	**100.00**	**98.33**	**100.00**	**100.00**	

Test Results

Contract: DominationToken

DominationToken Initializing Phase Test Cases

- ✓ should initialize name of domination token correctly (220ms)
- ✓ should initialize symbol of domination token correctly (244ms)
- ✓ should initialize default operators of domination token correctly (630ms)

DominationToken Functions Phase Test Cases

- ✓ should set transfers allowed correctly (551ms)
- ✓ shouldn't set transfers allowed if msg.sender hasn't role of TRANSFER_TOGGLER (1201ms)

Contract: Staking

Staking Initializing Phase Test Cases

- ✓ should initialize lpToken correctly (271ms)
- ✓ should initialize domToken correctly (171ms)
- ✓ should initialize start of staking correctly (201ms)
- ✓ should initialize maximum DOM to be distributed correctly (128ms)
- ✓ shouldn't initialize maximum DOM if maximum DOM isn't more then zero (659ms)
- ✓ shouldn't initialize start of staking if start less then timestamp (684ms)
- ✓ shouldn't initialize end vesting correctly if cliff more then end (703ms)

Staking Functions Phase Test Cases

- ✓ should stake correctly (967ms)
- ✓ shouldn't stake if amount is zero (504ms)
- ✓ shouldn't stake if isn't allowance from token (976ms)
- ✓ shouldn't stake if staking isn't allowed (1019ms)
- ✓ should stakeFor correctly (1862ms)
- ✓ shouldn't stakeFor if address is zero (501ms)
- ✓ should unstake correctly (1668ms)
- ✓ shouldn't unstake if amount is zero (1353ms)
- ✓ shouldn't unstake if amount of unstake more then stake (1239ms)
- ✓ should view info about stake correctly (7488ms)
- ✓ should view info about rewards correctly (5398ms)
- ✓ should unstake without rewards correctly (2015ms)
- ✓ should get balance of contract correctly (2450ms)
- ✓ shouldn't unstake if timestamp more then reward's period (2386ms)
- ✓ should get details about accounts if totalStaked is zero (359ms)
- ✓ should withdraw leftover correctly (6891ms)

Contract: Vester

Vester Initializing Phase Test Cases

- ✓ should initialize address of token for disburse correctly (110ms)
- ✓ should initialize address of recipient correctly (128ms)
- ✓ should initialize amount for disburse correctly (117ms)
- ✓ should initialize start vesting correctly (105ms)
- ✓ should initialize cliff vesting correctly (268ms)
- ✓ should initialize end vesting correctly (293ms)
- ✓ should initialize time between withdrawals correctly (387ms)
- ✓ shouldn't initialize start vesting correctly if start less then timestamp (443ms)
- ✓ shouldn't initialize cliff vesting correctly if start more then cliff (569ms)
- ✓ shouldn't initialize end vesting correctly if cliff more then end (626ms)

Vester Functions Phase Test Cases

- ✓ should set recipient correctly (730ms)
- ✓ shouldn't set recipient if recipient is caller (502ms)
- ✓ shouldn't set recipient if address is zero (422ms)
- ✓ should claim correctly (1865ms)
- ✓ should claim if vestingEnd less then timestamp (3513ms)
- ✓ shouldn't claim if no role is assigned (741ms)
- ✓ shouldn't claim if cliff more then timestamp (502ms)
- ✓ shouldn't claim if lastUpdate not equal to start of staking (1690ms)

Contract: VesterFactory

VesterFactory Functions Phase Test Cases

- ✓ should add address of recipient correctly (380ms)
- ✓ should add start vesting correctly (154ms)
- ✓ should add cliff vesting correctly (148ms)
- ✓ should add end vesting correctly (145ms)
- ✓ should add time between withdrawals correctly (140ms)

51 passing (5m)

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

ZOKYO.