Teneo

SMART CONTRACT AUDIT

ZOKYO.

Nov 23rd, 2021 | v. 1.0

PASS

Zokyo Security Team has concluded that these smart contracts pass security qualifications and bear no security or operational risk

TECHNICAL SUMMARY

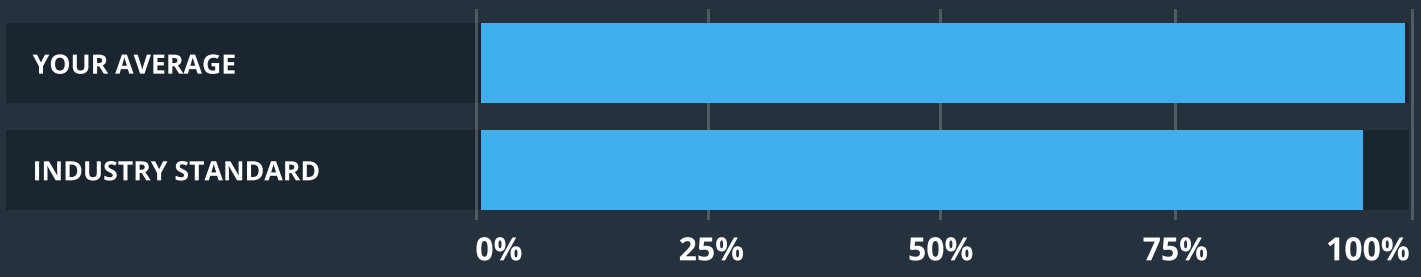
This document outlines the overall security of the Teneo Finance smart contracts, evaluated by Zokyo's Blockchain Security team.

The scope of this audit was to analyze and document the Teneo Finance 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 99.6%, 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 Teneo Finance 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 Teneo Finance repository.

Repository:
https://github.com/TeneoFinance/contracts/commit/a9749ab5e3aee310010ae2691bd8ccd820510145

Last commit:
874a94b645944749edd6f0b47ceefdc88ba56691

Contracts under the scope:

- AutomatedMarketMakerETH;
- AutomatedMarketMakerERC20;
- TenToken.

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 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.	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.

SUMMARY

The Zokyo team has conducted a security audit of the given codebase. The contracts provided for an audit are well written and structured. All the findings within the auditing process are presented in this document.

There were no critical issues found during the audit. We found 1 issue with a medium severity level and 2 informational issues. All of the mentioned findings were fixed by the Teneo team.

Based on the audit results the score is set to 99.

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

Incorrect check in decreaseAllowance() function at TenToken.sol

MEDIUM | RESOLVED

When calling the _allowances mapping in line 274, the indexes are confused, so we get a different result than we expect. This issue seriously affects the ability of the contract to decrease the allowance.

```
require(_allowances[sender][_msgSender()] >= amount, "transferFrom: transfer amount exceeds allowance");
```

Recommendation:
Replace _allowances[spender][_msgSender()] with _allowances[_msgSender()][spender].

Excessive check in _doTransfer() function at TenToken.sol

INFORMATIONAL | RESOLVED

In line 292 checks the condition _balanceOf (sender)> = amount but it is already checked in line 287.

Recommendation:
Remove one of them.

Incorrect revert message in setBuyAndSellFeeMultiplier() function at TenToken.sol

INFORMATIONAL | RESOLVED

In line 517 checks the condition bFeeDivisor > 0, but the message is about the sellFeeMultiplier variable.

```
require(bFeeDivisor > 0, "setBuyAndSellFeeMultiplier: sellFeeMultiplier has to be greater than zero.");
```

Recommendation:
Edit the message as follows: "setBuyAndSellFeeMultiplier: bFeeDivisor has to be greater than zero.".

	AutomatedMarketMakerETH	TenToken
Re-entrancy	Pass	Pass
Access Management Hierarchy	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass
Unexpected Ether	Pass	Pass
Delegatecall	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

	AutomatedMarketMakerERC20
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 team

As part of our work assisting Teneo Finance 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 Teneo Finance 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
amm\	100.00	81.25	100.00	100.00	
AutomatedMarketMakerERC20.sol	100.00	83.33	100.00	100.00	
AutomatedMarketMakerETH.sol	100.00	80.00	100.00	100.00	
token\	99.46	95.31	98.39	99.46	
TenToken.sol	99.46	95.31	98.39	99.46	742
All files	**99.60**	**92.50**	**98.70**	**99.60**	

Test Results

```
Contract: AutomatedMarketMakerETH
  Functions:
    toggleBuy
      ✓ should change buy possibility correctly (416ms)
    getTenToken
      ✓ should return the ten-Token address correctly (115ms)
    buy
```

✓ should mint new ten-Tokens for the sent ERC20 tokens correctly (778ms)
✓ should catch event
✓ should revert if the buy state is false (1265ms)
sell
✓ should burn ten-Tokens and send the ERC20 token to caller correctly (1392ms)
✓ should catch event
✓ should revert if the token amount exceeds the contract balance (299ms)
getInPrice
✓ should return the actual price for swapping pegged token to ten-Token correctly (223ms)
getOutPrice
✓ should return the actual price for swapping ten-Token to pegged token correctly (240ms)

Contract: AutomatedMarketMakerETH
Functions:
toggleBuy
✓ should change buy possibility correctly (410ms)
buy
✓ should mint new ten-Tokens for the sent ETH tokens correctly (591ms)
✓ should catch event
✓ should revert if the sent value and amount are not the same (292ms)
✓ should revert if buy state is false (553ms)
sell
✓ should burn ten-Tokens and send the ETH to the caller correctly (1282ms)
✓ should catch event
✓ should revert if the token amount exceeds contract balance (299ms)
getInPrice
✓ should return the actual price for swapping pegged token to ten-Token correctly (226ms)
getOutPrice
✓ should return the actual price for swapping ten-Token to pegged token correctly (206ms)

Contract: TenToken
Functions:
initialize
✓ should set token name correctly (113ms)
✓ should set token symbol correctly (125ms)
✓ should set pegged decimals correctly (113ms)
✓ should set decimals correctly (80ms)
✓ should set calc decimals correctly (108ms)
totalSupply

```
    ✓ should return total supply correctly (162ms)
nettoSupply
    ✓ should return netto supply correctly (164ms)
nettoBalanceOf
    ✓ should return netto balance of account correctly (96ms)
approve/allowance
    ✓ should approve tokens correctly (424ms)
    ✓ should catch Approval event
    ✓ should revert if approve to the zero address (271ms)
transferFrom
    ✓ should transfer tokens correctly (1161ms)
    ✓ should catch event
    ✓ should revert if transfer amount exceeds allowance (1137ms)
    ✓ should revert if transfer from the zero address (259ms)
increaseAllowance
    ✓ should increase allowance for spender correctly (550ms)
decreaseAllowance
    ✓ should decrease allowance for spender correctly (364ms)
    ✓ should revert if subtracted amount is greater than current allowance (317ms)
blacklistAccount
    ✓ should blacklists an account correctly (1467ms)
    ✓ should put the account balance in the feereserve correctly (190ms)
transfer
    ✓ should transfer ten-Tokens correctly (797ms)
    ✓ should catch event
    ✓ should revert if transfer amount exceeds sender`s balance (255ms)
    ✓ should revert if recipient is a zero address (244ms)
    ✓ should revert if transfer amount is a zero (209ms)
    ✓ should revert if recipient is blacklisted (254ms)
isBlacklisted
    ✓ should return true if an account is blacklisted (109ms)
    else
      ✓ should return false (145ms)
getMultiplierOf
    ✓ should return the fee account multiplier correctly (109ms)
globalMultiplier
    ✓ should return global multiplier correctly (114ms)
globalMultiplier
    ✓ should return global multiplier correctly (109ms)
```

getFeeDivisor
  ✓ should return fee divisor correctly (118ms)
getFees
  ✓ should return fee correctly (109ms)
getDaoTxFee
  ✓ should return dao tx fee correctly (87ms)
getActualFees
  ✓ should return actual fees correctly (148ms)
getActualBuyFees
  ✓ should return actual buy fees correctly (124ms)
getActualSellFees
  ✓ should return actual sell fees correctly (91ms)
buyFeeMultiplier
  ✓ should return buy fee multiplier correctly (126ms)
buyFeeDivisor
  ✓ should return buy fee divisor correctly (109ms)
sellFeeMultiplier
  ✓ should return sell fee multiplier correctly (98ms)
sellFeeDivisor
  ✓ should return sell fee divisor correctly (224ms)
feeReserve
  ✓ should return fee reserve correctly (119ms)
getReflowPer
  ✓ should return reflowPer correctly (103ms)
ammOf
  ✓ should return AMM address correctly (86ms)
setFees
  ✓ should set fee correctly (573ms)
  ✓ should revert if new fee is greater than max (200ms)
setDaoTxFee
  ✓ should set dao tx fee correctly (482ms)
  ✓ should revert if the new dao fee is greater than max (284ms)
setBuyAndSellFeeMultiplier
  ✓ should set buy and sell calculation of the fees correctly (1446ms)
  ✓ should revert if new sellFeeDivisor is zero (225ms)
  ✓ should revert if new buyFeeDivisor is zero (346ms)
setDaoWallet
  ✓ should set dao wallet correctly (531ms)
setAMM

```
    ✓ should set amm address correctly (1356ms)
    ✓ should revert if amm address is already set (234ms)
  compoundWallet
    ✓ should compound wallet with reflow correctly (1314ms)
  doReflow
    ✓ should do reflow correctly (1114ms)
  fountain
    ✓ should fountain amount correctly (853ms)
    ✓ should revert if transfer amount exceeds balance (217ms)
  mint
    ✓ should mint tokens correctly (753ms)
    ✓ should catch event
    ✓ should revert if the mint amount is zero (239ms)
    ✓ should revert if caller is not the AMM (322ms)
    ✓ should revert if the recipient is zero address (326ms)
  burn
    ✓ should burn tokens correctly (1228ms)
    ✓ should revert if caller is not the AMM (258ms)
    ✓ should revert if burn amount is zero (226ms)
    ✓ should revert if burn from the zero address (236ms)
    ✓ should revert if burn amount exceeds balance (288ms)
  addSyncAdr
    ✓ should add sync address correctly (889ms)
  getSyncAdrLength
    ✓ should return sync array length correctly (111ms)
  findAdr
    ✓ should return the index for given address correctly (203ms)
    ✓ should revert if the address is not found (116ms)
  removeIndex
    ✓ should remove the address by index correctly (394ms)
    ✓ should revert if the index is greater than sync array length (286ms)
    ✓ should revert if sync array is empty (493ms)
  removeAddress
    ✓ should remove the address correctly (1559ms)

  96 passing (49s)
```

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

ZOKYO.