TheStandard.io

SMART CONTRACT AUDIT

ZOKYO.

October 13th, 2022 | v. 1.0

PASS

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

TECHNICAL SUMMARY

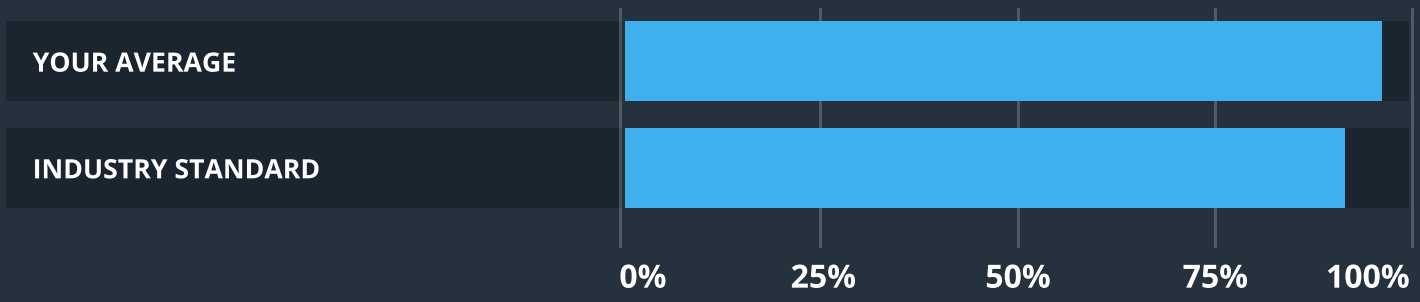
This document outlines the overall security of the TheStandard.io smart contracts, evaluated by Zokyo's Blockchain Security team.

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

Contract Status

There were 0 critical issues found during the audit. (See Complete Analysis)

Testable Code



The testable code is 99,22%, which is above the industry standard of 95%. (See Complete Analysis)

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 TheStandard.io 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 TheStandard.io repository.

Repository - https://github.com/the-standard/ibco/tree/master/contracts

Last commit - 8eecddc3c905659cd1afb1169fe156233db695d1

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

- BondingCurve.sol
- TokenManager.sol
- SEuroCalculator.sol
- SEuroOffering.sol

Throughout the review process, Zokyo Security ensures 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 in efficient use of resources, without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the latest vulnerabilities;
- Meets best practices in code readability, etc.

Zokyo's Security Team has followed best practices and industry-standard techniques to verify the implementation of TheStandard.io 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.

EXECUTIVE SUMMARY

Zokyo auditing team has run a deep investigation of given code. The contracts are in good condition, well written and structured.

During the auditing process, there were some issues with medium and low severity and informational issues found. After the technical review of fixes from TheStandard.io team, we can state issues are resolved and unresolved in the doc accordingly.

All the mentioned findings may have an effect only in case of specific conditions performed by the contract owner and the investors interacting with it.

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. Acknowledged means that the issue was sent to the client team and the client team are aware of it, but they have chosen to not solved it. 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 it can lead to a significant loss, funds may be lost or allocated incorrectly.

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

LOW | RESOLVED

TokenManager.sol addAcceptedToken function allows the owner to add the same token multiple times.

Recommendation:
Allow only unique tokens to be added to the accepted token symbols.

LOW | RESOLVED

TokenManager.sol Removing an accepted token that is added by the owner multiple times is possibly not removed completely in a single transaction.
Example flow:

- Initialize contract
- Add a token e.g, “usdt” 3 times.
- Call `getAcceptedTokens. Result: ['weth','usdt', 'usdt', 'usdt']`
- Remove the token e.g. ”usdt” 1 time
- Call `getAcceptedTokens Result: ['weth','usdt']`

Recommendation:
Fix the logic in removeAcceptedToken such that the loop breaks
when a token is found if only unique token symbols are permitted. Alternatively, use enumerables (see: https://docs.openzeppelin.com/contracts/3.x/api/utils.)

INFORMATIONAL | UNRESOLVED

TokenManager.sol Possible Gas optimization in the deleteToken function.
Recommendation: Storing array length in `len` saves gas rather than reading the length every time. Also, `++i` saves gas rather than `i++`. The unchecked keyword can be used while incrementing if the number of tokens to be added is intended to exceed the maximum value of uint256 type.
Reference video:
https://www.youtube.com/watch?v=4r20M9Fr8lY&ab_channel=SmartContractProgrammer

Current code:

```
for (uint256 i = index; i < tokenSymbols.length - 1; i++)
```

Suggestion:

```
uint256 len = tokenSymbols.length;

for (uint256 i = index; i < len; ) {
    ...
    unchecked {
        ++i;
    }
}
```

INFORMATIONAL | UNRESOLVED

TokenManager.sol DOS - Block Gas Limit: For large array lengths, the gas usage will be significantly higher and can lead to the failure of a transaction to be included in the blockchain due to Block Gas Limit https://swcregistry.io/docs/SWC-128

INFORMATIONAL | UNRESOLVED

TokenManager.sol The deployer can put any address as the Oracle address. This address is used by other contracts in the project for making calculations. It is the responsibility of the deployer to put the correct oracle address.

MEDIUM | RESOLVED

SEuroOffering.sol - body of `transferCollateral()`: As this method is triggered by an external call of swap and `swapETH`. An amount of an asset token is meant to be transferred to `collateralWallet` in exchange for SEuro.

```
if (collateralWallet != address(0))
 _token.transfer(collateralWallet, _amount);
```

In case collateralWallet is zero-address token is not transferred to collateral wallet, hence token amount will be kept in balance of SEuroOffering contract. It is worth noting that the amount of token kept will be inaccessible. Hence, we end up having lost inaccessible assets. In biref: this leads to a scenario in which SEuroOffering can receive assets which are not going to be transferable. Assets shall not be accessible or controlled in this case, hence will be lost.

Recommendation
replace `if` by `require`.

MEDIUM | UNRESOLVED

SEuroOffering.sol - `swap()` and `swapETH()` do not enable caller to determine the minimum quantity of `seuros` to receive, which shall expose him/her to attacks. In this scenario, the caller might receive an amount of `seuros` less than s/he expects.

Recommendation
add argument to functions representing the minimum quantity user is willing to receive and a require statement to revert the transaction if that minimum quantity is not met.

fix-1: These functions are exposed to losing funds due to attacks or even accidental price change. Mainly it is getSeuros which is used to calculate price of the asset and determining how much to mint in return (i.e. it also includes oracle call). An unlikely and undesired can be calculated during this transaction, a check needed to be added so that the user determine the minimum amount of token he would like to receive in return.

MEDIUM | UNRESOLVED

SEuroOffering.sol - `transferCollateral` - function is transferring assets (i.e. tokens) without safeguarding that transfer and not validating return.

Recommendation
It is preferred to use `safeTransfer` while transferring ERC20.

LOW | UNRESOLVED

SEuroOffering.sol, in `setCollateralWallet(address)` : input address is not asserted to be non-zero address.

Recommendation
`require` statement

LOW | RESOLVED

SEuroOffering.sol - `body of swap(...) & swapETH()` : input `_amount` and `msg.value` are not required here to be non-zero leading to unnecessary computation.

INFORMATIONAL | UNRESOLVED

All contracts - Admin enjoys too much authority. The general theme of the code is that admin has power to call several functions like `setBondingCurve, setCalculator, setTokenManager` and more importantly granting roles in all the contracts which leads to enable him to manipulate funds. Some functions can be more highly severe to be left out controlled by one wallet more than other functions. Recommendation Apply governance / use multisig wallets.

INFORMATIONAL | RESOLVED

Lock the pragma to a specific version, since not all the EVM compiler versions support all the features, especially the latest ones 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
fix version to 0.8.15

INFORMATIONAL | UNRESOLVED

SEuroCalculator.sol - Reference of `BondingCurve, TokenManager , SEuroCalculator.` Generally referencing the contracts as is rather than implementing interfaces.

Recommendation
implement interfaces for these contracts to be referenced.

INFORMATIONAL | UNRESOLVED

BondingCurve.sol - in body of `getBucketPrice(): getBucketMidpoint(_bucketIndex)` is supposed to represent x in this formula: `y = k * (x / m)^j + i`, where `x =` current total supply of sEURO by Bonding Curve according to doc provided in code which is `ibcoTotalSupply`. But we have `getBucketMidpoint(_bucketIndex) = (_bucketIndex * bucketSize) + (bucketSize / 2)` which is kind of a quantized quantity of `ibcoTotalSupply` and not the same.

fix-1: No change done by devs here to address the issue. The developing team might tell us that this does not harm the tokenomics of the project. Based upon their knowledge about the tokenomics the issue shall be considered irrelevant from our side.

	BondingCurve.sol	SEuroCalculator.sol
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

	SEuroOffering.sol	TokenManager.sol
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

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Zokyo Security team

As part of our work assisting TheStandard.io 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 TheStandard.io contract requirements for details about issuance amounts and how the system handles these.

```
BondingCurve.sol
    Initialization
    ✓ Tests initialization values and roles (44ms)
    calculatePrice
    ✓ calculatePrice (205ms)
    updateCurrentBucket
    ✓ updateCurrentBucket (86ms)
    readOnlyCalculatePrice
    ✓ readOnlyCalculatePrice
 SEuroCalculator
    Initialization
    0xb49f677943BC038e9857d61E7d053CaA2C1734C18
    0x6b7aacb44c48077765b60691bfe23f16a280e4728976c80332b6e45a0d8ee138
    0x8bCe54ff8aB45CB075b044AE117b8fD91F9351aB
    ✓ Initialization
    setBondingCurve()
    ✓ setBondingCurve (52ms)
    readOnlyCalculate()
    ✓ readOnlyCalculate (92ms)
SEuroOffering
    Initialization
    ✓ Initialization
    activate()
    ✓ activate
    setCollateralWallet()
    ✓ setCollateralWallet
    complete()
    ✓ complete (64ms)
    setBondingCurve()
    ✓ setBondingCurve
```

```
    setTokenManager()
    ✓ setTokenManager()
    setCalculator()
    ✓ setCalculator
    swap()
    ✓ swaps for token (268ms)
    swapETH
    ✓ swaps for eth (198ms)
    ✓ should update the price in bonding curve when bucket is crossed (617ms)
    readOnlyCalculateSwap
    ✓ readOnlyCalculateSwap (93ms)
TokenManager
    ✓ gets list of accepted tokens (1379ms)
    ✓ gets token details by name
    ✓ reverts if requested token does not exist
    adding tokens
    ✓ allows owner to add new token (56ms)
    ✓ does not allow non-owner to add token
    removing tokens
    ✓ allows owner to remove new token (59ms)
    ✓ does not allow non-owner to remove token (53ms)


25 passing (18s)
```

Code coverage

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

FILE	% STMTS	% BRANCH	% FUNCS	% LINES	% Uncovered Lines
BondingCurve.sol	96.88	94.44	100	100	
SEuroOffering.sol	100	86.96	100	100	
SEuroCalculator.sol	100	80	100	100	
TokenManager.sol	100	85.71	100	100	
All files	**99.22**	**86.77**	**100**	**100**	

We are grateful to have been given the opportunity to work with the TheStandard.io 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's Security Team recommends that the TheStandard.io team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

ZOKYO.