HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: TOMB
Date: March 29th, 2022

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Document

Name	Smart Contract Code Review and Security Analysis Report for TOMB.
Approved By	Evgeniy Bezuglyi | SC Department Head at Hacken OU
Type of Contracts	ERC20 token
Platform	EVM
Language	Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Website	https://tomb.finance
Timeline	10.03.2022 - 29.03.2022
Changelog	25.03.2022 - Initial Review 29.03.2022 - Revise

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Table of contents

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Introduction

Hacken OÜ (Consultant) was contracted by TOMB (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contracts.

Scope

The scope of the project is smart contracts in the repository:

Repository:
https://github.com/tombfinance/tombfinance-contracts
Commit:
c7d44f87cb64878277bd737945e6feaf4f6f9516
Technical Documentation: No (https://docs.tomb.finance/)
JS tests: No
Contracts:
interfaces/IOracle
owner/Operator.sol
lib/SafeMath8.sol
Tomb.sol

We have scanned this smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that are considered:

Category	Check Item
Code review	▪ Reentrancy ▪ Ownership Takeover ▪ Timestamp Dependence ▪ Gas Limit and Loops ▪ Transaction-Ordering Dependence ▪ Style guide violation ▪ EIP standards violation ▪ Unchecked external call ▪ Unchecked math ▪ Unsafe type inference ▪ Implicit visibility level ▪ Deployment Consistency ▪ Repository Consistency
Functional review	▪ Business Logics Review ▪ Functionality Checks ▪ Access Control & Authorization ▪ Escrow manipulation ▪ Token Supply manipulation ▪ Assets integrity ▪ User Balances manipulation ▪ Data Consistency ▪ Kill-Switch Mechanism

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Executive Summary

The score measurements details can be found in the corresponding section of the methodology.

Documentation quality

The Customer provided user-friendly docs but neither functional requirements nor technical requirements. However, while it is a pretty straightforward ERC20 token contract with some additions, it is mostly self-documented.
The total Documentation Quality score is **7** out of **10**.

Code quality

The total CodeQuality score is **5** out of **10**. Code duplications. No unit tests were provided. Not following solidity code style guidelines. No NatSpec blocks. Contradictions in the code.

Architecture quality

The architecture quality score is **7** out of **10**. Outdated solidity compiler used. Token logic is in one file.

Security score

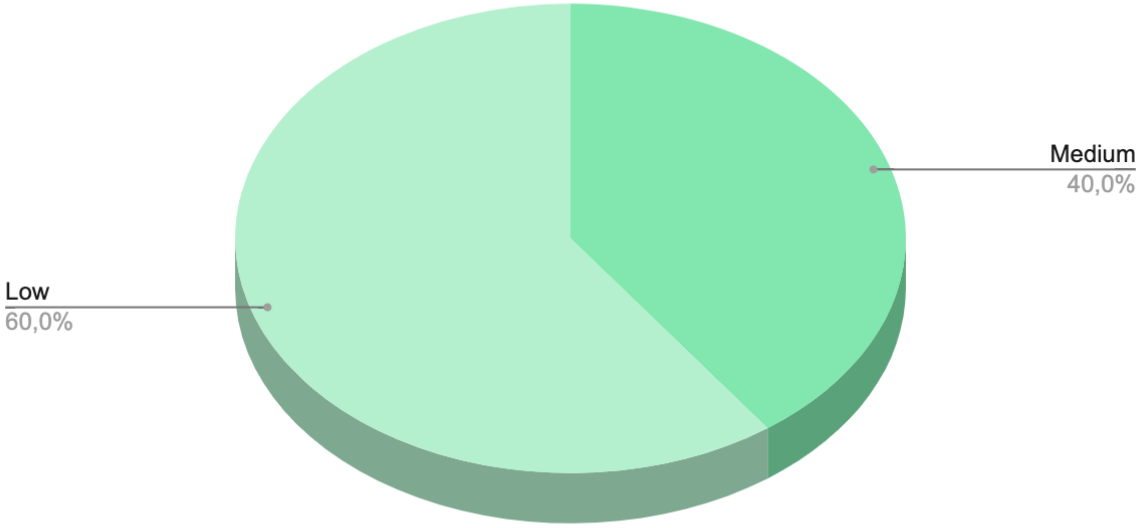
As a result of the audit, security engineers found **2** medium and **3** low severity issues. The security score is **10** out of **10**. All found issues are displayed in the “Issues overview” section.

Summary

According to the assessment, the Customer's smart contract has the following score: **8.9**

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Graph 1. The distribution of vulnerabilities after the audit.



Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to assets loss or data manipulations.
High	High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g., public access to crucial functions
Medium	Medium-level vulnerabilities are important to fix; however, they cannot lead to assets loss or data manipulations.
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that cannot have a significant impact on execution

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Findings

Critical

No critical severity issues were found.

High

No high severity issues were found.

Medium

1. **Tautology or contradiction.**

   Functions check that `_index` should be greater than or equal to zero. However, the variable type is `uint8`, which stands for “unsigned integer 8 bits”. Looking through the type, the unsigned integer, by its definition, could not be less than zero. Therefore, this statement does not make any sense and will never work.

   The revert message, “Index has to be higher than 0”, looks like the transaction should be reverted when the `_index` is equal to zero, but that will never happen.

   Contract: Tomb.sol

   Functions: setTaxTiersTwap, setTaxTiersRate, _updateTaxRate

   Recommendation: while it’s already deployed, no recommendations here

   Status: Acknowledged

2. **No tests were provided.**

   It is highly recommended to add tests for the contract's code. Unit tests would help ensure functions are working properly, while integration tests would ensure contracts are working perfectly.

   Scope: testing

   Recommendation: create unit and integration tests covering up to 100% statements and branches.

   Status: Acknowledged

Low

1. **The solidity version is outdated.**

   The solidity compiler version used in contracts is outdated (0.6.12 vs. 0.8.13). Using outdated contracts may lead to old bugs that were already fixed in the recent versions. Recent versions are more optimized in gas usage and have additional pros.

   Recommendation: while it is already deployed, there are no recommendations here

   Status: Acknowledged

2. **A public function that could be declared external.**

Public functions that are never called by the contract should be declared **external**.

Contracts: Tomb.sol, Operator.sol

Functions: Tomb.isAddressExcluded, Tomb.setTaxTiersTwap, Tomb.setTaxTiersRate, Tomb.setTaxOffice, Tomb.setBurnThreshold, Tomb.enableAutoCalculateTax, Tomb.mint, Tomb.disableAutoCalculateTax, Tomb.setTombOracle, Tomb.setTaxCollectorAddress, Tomb.setTaxRate, Tomb.includeAddress, Operator.transferOperator, Operator.operator, Operator.isOperator

Recommendation: while it is already deployed, there are no recommendations here

Status: Acknowledged

3. **No return statement.**

The function defines a return type as boolean but never explicitly returns any value. That means that the return value would always be **false**, which could be unexpected by the caller.

Contract: Tomb.sol

Function: setBurnThreshold

Recommendation: while it is already deployed, there are no recommendations here

Status: Acknowledged

Recommendations

1. Write unit and integration test.
2. Write clear and clean technical documentation (diagrams, flows, sequences, tech specs).
3. Use a recent solidity compiler version.
4. Follow the latest code style guides and the best practices.
5. Check and test the code before deploying it to production.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed by the best industry practices at the date of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The audit makes no statements or warranties on the security of the code. It also cannot be considered a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the audit cannot guarantee the explicit security of the audited smart contracts.