SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Wombat Exchange
Date: April 20th, 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 Wombat Exchange.
Approved By	Evgeniy Bezuglyi | SC Department Head at Hacken OU
Type of Contracts	ERC20 token; Exchange; Staking
Platform	EVM
Language	Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Website	https://wombat.exchange
Timeline	10.03.2022 - 20.04.2022
Changelog	24.03.2022 - Initial Review 04.04.2022 - Revise 20.04.2022 - Revise

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by Wombat Exchange (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/wombat-exchange/wombat
Commit:
    8a0e9cb9806f8d3ba7df96bf07f6e61a65389c4b
Technical Documentation: Yes
(https://www.wombat.exchange/Wombat_Whitepaper_Public.pdf; README.md)
JS tests: Yes (test)
Contracts:
    wombat-governance/libraries/DSMath.sol
    wombat-core/pool/Pool.sol
    wombat-governance/MasterWombat.sol
    wombat-core/libraries/SafeCast.sol
    wombat-core/asset/AggregateAccount.sol
    wombat-peripheral/token/WombatERC20.sol
    wombat-governance/VeERC20Upgradeable.sol
    wombat-core/pool/CoreV2.sol
    wombat-governance/Whitelist.sol
    wombat-peripheral/vesting/TokenVesting.sol
    wombat-governance/libraries/LogExpMath.sol
    wombat-core/libraries/SignedSafeMath.sol
    wombat-core/asset/Asset.sol
    wombat-governance/VeWom.sol
    wombat-core/pool/PausableAssets.sol
    wombat-core/libraries/DSMath.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

Executive Summary

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

Documentation quality

The Customer provided a Whitepaper and technical documentation with a high-level design diagram. However, no flows, sequences, or tech specs were provided. The total Documentation Quality score is **7** out of **10**.

Code quality

The total CodeQuality score is **8** out of **10**. Code duplications. Unit tests were provided. Good NatSpec coverage. Lots of useful comments.

Architecture quality

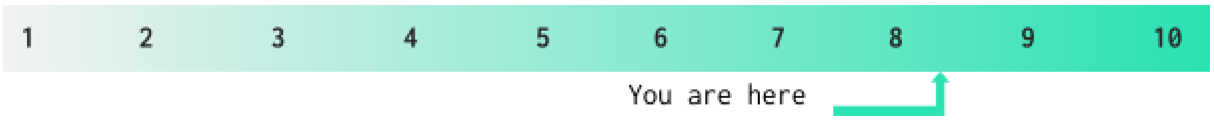
The architecture quality score is **7** out of **10**. Logic is split across files. Functions are overwhelmed with template code that could be moved to separate functions and be reused.

Security score

As a result of the audit, security engineers found **1** low severity issue. The security score is **10** out of **10**. All found issues are displayed in the "Issues overview" section.

Summary

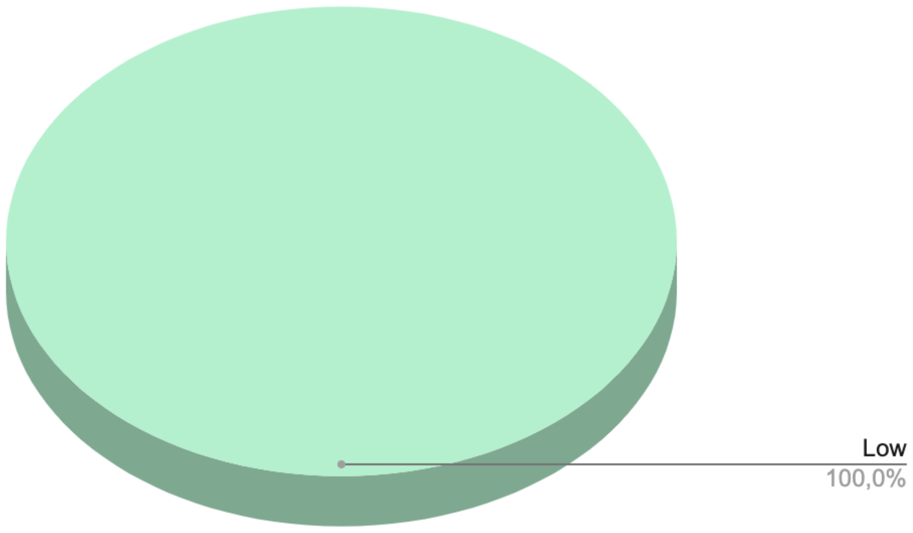
According to the assessment, the Customer's smart contract has the following score: **8.2**



Notices

1. The owner can withdraw all rewards (wom) from the MasterWombat contract.

Graph 1. The distribution of vulnerabilities after the audit.



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

Findings

■■■■ Critical

No critical severity issues were found.

■■■ High

No high severity issues were found.

■■ Medium

No medium severity issues were found.

■ Low

1. **A declaration shadows an existing declaration.**

   Shadowed declarations could make a code not obvious and may lead to inconsistencies.

   - **uint256 haircut** @Pool.sol#719
     shadows the return variable @Pool.sol:728
   - return variable **exchangeRate** @Pool.sol#745
     shadows the function @Pool.sol:745
   - **uint256 tipBucketBalance** @Pool.sol#770
     shadows the function @Pool.sol:757
   - **uint256 tipBucketBalance** @Pool.sol#789
     shadows the function @Pool.sol:757

   Contract: Pool.sol

   Recommendation: rename corresponding variables.

   Status: Fixed (Revised Commit: f3349a4)

2. **Interface function virtual declaration.**

   All interface functions are implicitly “virtual”. No need to add a corresponding keyword.

   Interface: IAsset.sol

   Function: underlyingTokenDecimals

   Recommendation: remove the keyword “virtual”.

   Status: Fixed (Revised Commit: f3349a4)

3. **Constructor visibility.**

   Starting solidity compiler version 0.7 there is no need to declare the constructor visibility anymore because it is ignored.

   Contract: WombatERC20.sol

   Recommendation: remove constructor visibility.

   Status: Fixed (Revised Commit: f3349a4)

4. **Unused function parameter.**

   Function parameter `to` is not used in the code.

   Contract: Pool.sol

   Function: _swap

   Recommendation: remove the declaration of the parameter. If it needs the consistency parameter, remove just the name while leaving the type declaration.

   Status: Fixed (Revised Commit: f3349a4)

5. **A function could be declared as `view`.**

   The function that does not update any state should be declared as `view` to save gas.

   Contract: PausableAssets.sol

   Function: requireAssetPaused

   Recommendation: add the keyword `view` to the function declaration.

   Status: Fixed (Revised Commit: f3349a4)

6. **A function could be declared as `pure`.**

   The function that neither updates nor access any state should be declared as `pure` to save gas.

   Contracts: Pool.sol, VeWom.sol

   Functions: _checkLiquidity, _checkAddress, _checkSameAddress, _checkAmount, _expectedVeWomAmount

   Recommendation: add the keyword `view` to the function declaration.

   Status: Only the `_expectedVeWomAmount` was fixed

7. **Unused state variable.**

   The internal state variable declared in the contract is never used in the contract itself.

   Contracts: Asset.sol

   Variable: reserved

   Recommendation: remove unused variable.

   Status: Fixed (Revised Commit: f3349a4)

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

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

   Contracts: MasterWombat.sol, VeERC20Upgradeable.sol, VeWom.sol

Variable: MasterWombat.add, MasterWombat.set, MasterWombat.emergencyWithdraw, VeERC20Upgradeable.name, VeERC20Upgradeable.symbol, VeERC20Upgradeable.decimals, VeWom.initialize

Recommendation: use the **external** attribute for functions never called from the contract.

Status: Fixed (Revised Commit: f3349a4)

9. **Using library.**

While it is declared to use a SignedSafeMath for int256, there is no need to call the SignedSafeMath implicitly in the code. It is recommended to write the call as a direct function call from the variable itself.

Contracts: CoreV2.sol

Functions: _solveQuad, exactDepositLiquidityInEquilImpl, withdrawalAmountInEquilImpl

Recommendation: call the function from the int256 variable directly (i.e.: **l.sqrt(b)** instead of SignedSafeMath.sqrt(l, b)).

Status: Fixed (Revised Commit: 8a0e9cb)

Recommendations

1. Follow solidity code style guidelines.
2. It is good to have more tech docs.

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.