HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Wombat
Date: Aug 18th, 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
Approved By	Noah Jelich | Senior Solidity SC Auditor at Hacken OU
Type	Staking
Platform	EVM
Network	Ethereum, Polygon
Language	Solidity
Methods	Manual Review, Automated Review, Architecture Review
Website	https://www.wombat.app
Timeline	08.08.2022 - 18.08.2022
Changelog	15.08.2022 - Initial Review 18.08.2022 - Second Review

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 Wombat (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:

Initial review scope

Repository:
https://github.com/wombat-tech/wombat-token/tree/staking
Commit:
c637fe87730160e10510ebdd56bd2b6bdb129723
Documentation:
Whitepaper

Integration and Unit Tests: Yes
Contracts:
File: ./contracts/WombatStaking.sol
SHA3: 2a24c49b344049f301491a55863d283ed734307d393422e8e63562d5d23a3526

Second review scope

Repository:
https://github.com/wombat-tech/wombat-token/tree/staking
Commit:
5a985155e133456a564da08e9049755a2dd520a2
Documentation:
Whitepaper

Integration and Unit Tests: Yes
Contracts:
File: ./contracts/WombatStaking.sol
SHA3: ddf03183d614bfa1efe04c0da2dcd6a342fce0a5e53ee3b92fc698170b32326b

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Low	Low-level vulnerabilities are mostly related to outdated, unused, etc.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

Executive Summary

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

Documentation quality

The total Documentation Quality score is **10** out of **10**. Information about how to run the project and functional requirements were provided.

Code quality

The total CodeQuality score is **9** out of **10**. Deployment and basic user interactions are covered with tests. **Branch test coverage is 95.45%.** Missing one negative test case.

Architecture quality

The architecture quality score is **10** out of **10**. Code follows a single responsibility principle and is well-organized.

Security score

As a result of the audit, the code does not contain any issues. The security score is **10** out of **10**.

All found issues are displayed in the “Findings” section.

Summary

According to the assessment, the Customer's smart contract has the following score: **9.9**.

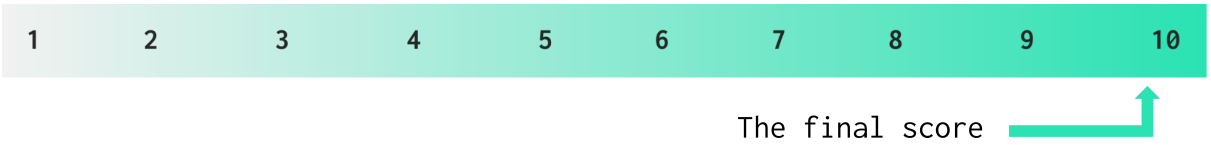

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
10 August 2022	2	1	1	0
17 August 2022	0	0	0	0

Checked Items

We have audited provided smart contracts for commonly known and more specific vulnerabilities. Here are some of the items that are considered:

Item	Type	Description	Status
Default Visibility	SWC-100 SWC-108	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed
Integer Overflow and Underflow	SWC-101	If unchecked math is used, all math operations should be safe from overflows and underflows.	Not Relevant
Outdated Compiler Version	SWC-102	It is recommended to use a recent version of the Solidity compiler.	Passed
Floating Pragma	SWC-103	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Passed
Unchecked Call Return Value	SWC-104	The return value of a message call should be checked.	Passed
Access Control & Authorization	CWE-284	Ownership takeover should not be possible. All crucial functions should be protected. Users could not affect data that belongs to other users.	Passed
SELFDESTRUCT Instruction	SWC-106	The contract should not be self-destructible while it has funds belonging to users.	Passed
Check-Effect-Interaction	SWC-107	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed
Assert Violation	SWC-110	Properly functioning code should never reach a failing assert statement.	Passed
Deprecated Solidity Functions	SWC-111	Deprecated built-in functions should never be used.	Passed
Delegatecall to Untrusted Callee	SWC-112	Delegatecalls should only be allowed to trusted addresses.	Not Relevant
DoS (Denial of Service)	SWC-113 SWC-128	Execution of the code should never be blocked by a specific contract state unless it is required.	Passed
Race Conditions	SWC-114	Race Conditions and Transactions Order Dependency should not be possible.	Passed
Authorization through tx.origin	SWC-115	tx.origin should not be used for authorization.	Not Relevant
Block values as	SWC-116	Block numbers should not be used for time	Passed

a proxy for time		calculations.	
Signature Unique Id	SWC-117 SWC-121 SWC-122 EIP-155	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifier should always be used. All parameters from the signature should be used in signer recovery	Not Relevant
Shadowing State Variable	SWC-119	State variables should not be shadowed.	Passed
Weak Sources of Randomness	SWC-120	Random values should never be generated from Chain Attributes or be predictable.	Not Relevant
Incorrect Inheritance Order	SWC-125	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order.	Not Relevant
Calls Only to Trusted Addresses	EEA-Level-2 SWC-126	All external calls should be performed only to trusted addresses.	Not Relevant
Presence of unused variables	SWC-131	The code should not contain unused variables if this is not justified by design.	Passed
EIP standards violation	EIP	EIP standards should not be violated.	Passed
Assets integrity	**Custom**	Funds are protected and cannot be withdrawn without proper permissions.	Passed
User Balances manipulation	**Custom**	Contract owners or any other third party should not be able to access funds belonging to users.	Passed
Data Consistency	**Custom**	Smart contract data should be consistent all over the data flow.	Passed
Flashloan Attack	**Custom**	When working with exchange rates, they should be received from a trusted source and not be vulnerable to short-term rate changes that can be achieved by using flash loans. Oracles should be used.	Not Relevant
Token Supply manipulation	**Custom**	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the customer.	Not Relevant
Gas Limit and Loops	**Custom**	Transaction execution costs should not depend dramatically on the amount of data stored on the contract. There should not be any cases when execution fails due to the block Gas limit.	Passed
Style guide violation	**Custom**	Style guides and best practices should be followed.	Passed
Requirements Compliance	**Custom**	The code should be compliant with the requirements provided by the Customer.	Passed
Environment	**Custom**	The project should contain a configured	Passed

Consistency		development environment with a comprehensive description of how to compile, build and deploy the code.	
Secure Oracles Usage	**Custom**	The code should have the ability to pause specific data feeds that it relies on. This should be done to protect a contract from compromised oracles.	Not Relevant
Tests Coverage	**Custom**	The code should be covered with unit tests. Test coverage should be 100%, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Passed
Stable Imports	**Custom**	The code should not reference draft contracts, that may be changed in the future.	Passed

System Overview

Wombat is a Web 3 Gaming Platform with the following contracts:

- *Staking* — a contract that allows users to lock and unlock the funds they have in the contract. There is no reward mechanism in the contract.

Privileged roles

- The owner of the *Staking* contract can pause and unpause the *stake* function.
- The owner of the *Staking* contract can update the *unstake* cooldown period. This will not affect users who are already waiting for the *unstake* cooldown period.

Risks

- If the user calls the *unstake* function twice in a row, the cooldown time will be reset. As a result, the cooldown of the last *unstake* operation will be valid.

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

Findings

Critical

No Critical severity issues were found.

High

1. **Requirement violation**

   The owner of the contract can change the unstake cooldown period as desired.

   An erroneous value given while changing will not affect the duration of existing users who are already waiting but will affect new unstakers cooldown permanently.

   As a result, an extreme input value could lead to permanent locking of user funds due to a large waiting period.

   Path: ./contracts/WombatStaking.sol : setUnstakeTime()

   Recommendation: Limit the possible cooldown period or make it a constant.

   Status: Fixed (5a985155e133456a564da08e9049755a2dd520a2)

Medium

1. **Unchecked transfer**

   ERC20 implementations should return boolean values in transfer functions. In case of an erroneous situation, those functions will not revert. Instead, they will continue execution and return false.

   Path: ./contracts/WombatStaking.sol : claim()

   Recommendation: Implement return value checks or use SafeERC20 implementations.

   Status: Fixed (5a985155e133456a564da08e9049755a2dd520a2)

Low

1. **Secrets pushed to version control**

   The *.env* file configuration is not used in the project; the API keys are directly hardcoded in the *hardhat.config.ts* file. This will lead to attackers accessing private information about the project.

   Path: ./hardhat.config.ts

   Recommendation: Remove all API keys and private keys from the hardhat.config.ts file. Create an *.env* document and import this private information from the *.env* document.

   Status: Fixed (5a985155e133456a564da08e9049755a2dd520a2)

2. Missing event emitting

Events for critical state changes should be emitted for tracking things off-chain.

Path: ./contracts/WombatStaking.sol : setUnstakeTime()

Recommendation: Create and emit related events.

Status: Fixed (5a985155e133456a564da08e9049755a2dd520a2)

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 report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted to and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and 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.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

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, Consultant cannot guarantee the explicit security of the audited smart contracts.