HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Wombat
Date: July 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
Approved By	Evgeniy Bezuglyi | SC Audits Department Head at Hacken OU
Type	ERC20 token; Staking
Platform	EVM
Network	Ethereum, Polygon
Language	Solidity
Methods	Manual Review, Automated Review, Architecture review
Website	https://wombat.app
Timeline	04.07.2022 - 20.07.2022
Changelog	07.07.2022 - Initial Review 20.07.2022 - 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
Commit:
b4b0e4ccb44fd1bd9f2e9603ca4e9bfea5b34d4a
Technical Documentation:
Type: Whitepaper (partial functional requirements provided)
https://whitepaper.wombat.app/

Type: Technical description
https://whitepaper.wombat.app/

Type: Functional requirements
https://whitepaper.wombat.app/

Integration and Unit Tests: Yes
Contracts:
File: ./contracts/LinearCheckpointDualBeneficiaryVestingWallet.sol
SHA3: 212c660ca9bad9bda9a6be6aefa53201018e662b4197186e3fbc1aa02edb2830

File: ./contracts/LinearCheckpointVesting.sol
SHA3: 9bbba33b21d029f12b02241aee06d46b7872f3eb8d4b45520751ae85479c1ad9

File: ./contracts/LinearCheckpointVestingWallet.sol
SHA3: 015b50be44ea8a33f55fe7f24dbd479d4a868eec9b41b720e8a8fd1e9171b5fa

File: ./contracts/PercentageCheckpointVestingWallet.sol
SHA3: d2ab145c0e1d4db64de73f347eb736da79513ddf51b3027981c1ceb892fb305b

File: ./contracts/WombatChildToken.sol
SHA3: 0e2cbe60843e7bc006af09d143ab8a2a7686631d67dd406e5e63d5269474eb2d

File: ./contracts/WombatToken.sol
SHA3: 1dee402f296f5d77a7529470ffb2cc4cea0721575eefc979cec8a23a60f4c4c3

Second review scope

Repository:
https://github.com/wombat-tech/wombat-token
Commit:
b4091c1e5015152aad8a2573bf85b26d87f39fcf
Technical Documentation:
Type: Whitepaper (partial functional requirements provided)
https://whitepaper.wombat.app/

Type: Technical description
https://whitepaper.wombat.app/

Type: Functional requirements
https://whitepaper.wombat.app/

Integration and Unit Tests: Yes
Contracts:

File: ./contracts/LinearCheckpointDualBeneficiaryVestingWallet.sol
SHA3: 8779d23ada374fd51ad19934e3dc1933f4bb35d7e288d715871d91c5f3384dad

File: ./contracts/LinearCheckpointVesting.sol
SHA3: 3c2a9e20576b9c092f127799aa30693d963a428494e2b3a0bb54a65a98bb1038

File: ./contracts/LinearCheckpointVestingWallet.sol
SHA3: 5f8c0cb2a38e742b8213ea1276d62cfb9f29dffa50277468cbd6978e06137ca5

File: ./contracts/PercentageCheckpointVestingWallet.sol
SHA3: 043a89ced55f63dce26985812773b97f9834618b5b258f24bddfb6aa82147cf1

File: ./contracts/WombatChildToken.sol
SHA3: e0a6b7d03e1fd9fd4c1475bbd3bd18e777c39f47e999c9f5a14f7387068b034d

File: ./contracts/WombatToken.sol
SHA3: 177d18e587c4e558506bf052fbf42c067c732577381852bd9e85ba5626b9043a

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 **7** out of **10**. Functional requirements are partially missed. A technical description is not provided.

Code quality

The total CodeQuality score is **8** out of **10**. Deployment and basic user interactions are covered with tests. Negative cases coverage exists, but interactions by multiple users are not tested thoroughly.

Architecture quality

The architecture quality score is **6** out of **10**. **5** contracts reference other contracts that can be changed in the future.

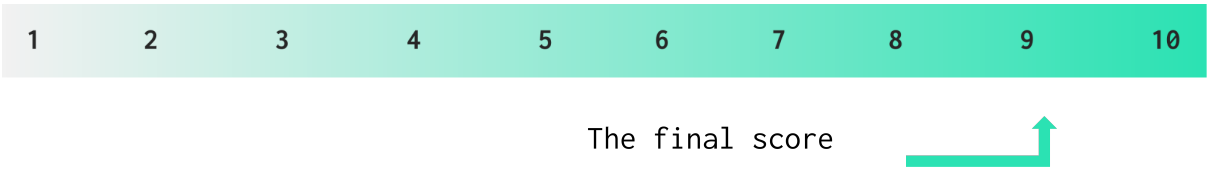
Security score

As a result of the second 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: **9.1**.



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

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.	Passed
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.	Not Relevant
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.	Not Relevant
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.	Passed
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	Passed
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.	Passed
Calls Only to Trusted Addresses	EEA-Level-2 SWC-126	All external calls should be performed only to trusted addresses.	Passed
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.	Not Relevant
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.	Passed
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 - The Web 3 Gaming Platform with the following contracts:

- *Wombat Token* — ERC-20 token that mints all initial supply to a deployer. Additional minting is not allowed.
  It has the following attributes:
  - Name: WOMBAT token
  - Symbol: WOMBAT
  - Decimals: 18
  - Total supply: 10b tokens.
  - Blockchain: Ethereum
- *Wombat Child Token* — ERC-20 Polygon token extended with the 'deposit' and 'withdraw' functions.
  Deposit - called by the ChildChainManagerProxy contract whenever a deposit is initiated from the root chain. This function internally mints the token on the child chain. In this case: Polygon.
  Withdraw - will internally burn the token on the child chain.
  It has the following attributes:
  - Name: WOMBAT token
  - Symbol: WOMBAT
  - Decimals: 18
  - Blockchain: Polygon
- *LinearCheckpointVestingWallet.sol* — is a smart contract for investors to lock their investment for a time and then release it according to a schedule. The schedule is a set of timestamps called checkpoints at which "chunks" of the investment in Wombat tokens become available. The lock-up period and the schedule are configured as timestamps during deployment. This allows to lock up tokens for e.g. 12 months and then release them over 6 months, 1/6 at a time.
- *PercentageCheckpointVestingWallet.sol* - is a smart contract to which community tokens are transferred and then can be claimed monthly. This contract allows us to lock up tokens and then release 2% monthly for 60 months. Afterward, the full remaining amount is releasable.
- *LinearCheckpointDualBeneficiaryVestingWallet.sol* - is a vesting wallet that releases to two addresses instead of one for tax reasons. Both addresses can initiate the release. This allows to lock up tokens for 24 months and then release 50% to one address and 50% to another address over the following 24 months 1/24 at a time.
- LinearCheckpointVesting.sol - is an abstract contract, which other contracts such as LinearCheckpointDualBeneficiaryVestingWallet and LinearCheckpointVestingWallet inherit. LinearCheckpointVesting contract contains the state variable _checkpoints that represent the timestamps (in seconds) at which chunks of the vested amount are released. LinearCheckpointVesting provides view functionality for vesting schedules.

Privileged roles

- The owner of the *WombatToken* contract will be the owner of all total token supply once WombatToken is deployed.
- *Pauser role* has the possibility to pause and unpause token transfers.
- *Two beneficiary addresses* - used *in* a vesting wallet that releases to these addresses instead of one for tax reasons. These two addresses will receive funds after the vesting period.

Risks

- In case of an admin keys leak, an attacker will be able to pause WombatToken contract at any moment, and it will lead to funds being frozen for any holders of tokens for some time. The DEFAULT_ADMIN_ROLE address will be able to unpause, but the attacker can pause WombatToken, at any time.

Findings

Critical

No high severity issues were found.

High

1. Unlimited token minting.

The mintable token amount should not exceed the amount declared in the documentation. This will lead to imbalances in tokenomics.

According to the documentation total token supply is 10 billion. However, the code has no upper limit for minting.

This will lead to imbalances in tokenomics.

File: ./contracts/WombatChildToken.sol

Contract: WombatChildToken

Function: mint

Recommendation: Limit the amount of tokens that can be minted according to the documentation.

Status: Fixed (Revised commit: b4091c1e5015152aad8a2573bf85b26d87f39fcf)

2. Undocumented behavior.

The code should not contain undocumented functionality. The existence of such functionality can lead to unexpected behavior of the contract.

The functionality allows the owner to pause all the token transfers anytime. Pausing functionality should be limited by clear contract rules. The documentation does not mention the functionality of transfers stopping.

This will lead to imbalances in tokenomics.

Files: ./contracts/WombatToken.sol

./contracts/WombatChildToken.sol

Contracts: WombatToken, WombatChildToken

Functions: pause, unpause

Recommendation: Remove pausing functionality or notify users about it in the provided documentation.

Status: Fixed (Revised commit: b4091c1e5015152aad8a2573bf85b26d87f39fcf)

Medium

1. Costly operations inside a loop.

Reading a state variable or an attribute of it may be costly in terms of Gas fees. The _checkpoints and _percentage variables are often used in a loop inside the _vestingSchedule function.

This can lead to high Gas consumption.

File: ./contracts/PercentageCheckpointVestingWallet.sol

Contract: PercentageCheckpointVestingWallet

State variables: _checkpoints, _percentage

Recommendation: Save the state variable or its attribute into a local variable and use it inside the loop.

Status: Fixed (Revised commit: b4091c1e5015152aad8a2573bf85b26d87f39fcf)

Low

1. Missing empty array check.

LinearCheckpointVesting's constructor does not contain empty array validation for 'checkpointTimestamps' parameter.

Missing an empty array check can lead to an error. For example the value of _checkpoints.length will be zero and it will lead to zero division error : (totalAllocation / _checkpoints.length) * currentCheckpoint;

File: ./contracts/LinearCheckpointVesting.sol

Contract: LinearCheckpointVesting

Recommendation: Implement empty array checks.

Status: Fixed (Revised commit: b4091c1e5015152aad8a2573bf85b26d87f39fcf)

2. Division before multiplication.

Solidity integer division might truncate.

LinearCheckpointDualBeneficiaryVestingWallet's 'release' function performs a multiplication on the result of a division:
uint256 mainBeneficiaryAmount = (releasable /100) *_percentageSplit;

As a result, performing multiplication before division can sometimes

avoid loss of precision.

File: ./contracts/LinearCheckpointDualBeneficiaryVestingWallet.sol

Contract: LinearCheckpointDualBeneficiaryVestingWallet

Function: release

Recommendation: Consider ordering multiplication before division.

Status: Fixed (Revised commit: b4091c1e5015152aad8a2573bf85b26d87f39fcf)

3. **Missing zero address validation.**

   Address parameters are being used without checking against the possibility of 0x0.

   This can lead to unwanted external calls to 0x0.

   File: ./contracts/LinearCheckpointDualBeneficiaryVestingWallet.sol

   Constructor variable: tokenAddress

   Recommendation: Implement zero address checks.

   Status: Fixed (Revised commit: b4091c1e5015152aad8a2573bf85b26d87f39fcf)

4. **Different pragma directives are used.**

   Usage of different pragma directives can lead to errors in code logic.

   LinearCheckpointDualBeneficiaryVestingWallet, LinearCheckpointVesting, LinearCheckpointVestingWallet, PercentageCheckpointVestingWallet, WombatChildToken, WombatToken use ^0.8.0 Solidity version.

   Files: ./contracts/LinearCheckpointDualBeneficiaryVestingWallet.sol

   ./contracts/LinearCheckpointVesting.sol

   ./contracts/LinearCheckpointVestingWallet.sol

   ./contracts/PercentageCheckpointVestingWallet.sol

   ./contracts/WombatChildToken.sol

   ./contracts/WombatToken.sol

   Contracts: LinearCheckpointDualBeneficiaryVestingWallet.sol, LinearCheckpointVesting.sol, LinearCheckpointVestingWallet.sol, PercentageCheckpointVestingWallet.sol, WombatChildToken.sol, WombatToken.sol

   Recommendation: Use a specific Solidity version for both contracts.

Status: Fixed (Revised commit: b4091c1e5015152aad8a2573bf85b26d87f39fcf)

5. **Functions that can be declared external.**

In order to save Gas, public functions that are never called in the contract should be declared as external.

Files: ./contracts/PercentageCheckpointVestingWallet.sol,

./contracts/LinearCheckpointVesting.sol

./contracts/LinearCheckpointDualBeneficiaryVestingWallet.sol

Contracts:PercentageCheckpointVestingWallet.sol
LinearCheckpointVesting.sol,
LinearCheckpointDualBeneficiaryVestingWallet.sol

Functions: checkpoints, start, duration, percentageSplit, release

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

Status: Fixed (Revised commit: b4091c1e5015152aad8a2573bf85b26d87f39fcf)

6. **Variable shadowing.**

State variables should not be shadowed.

The constructors of WombatChildToken and WombatToken shadow such variables as ‘name’ and ‘symbol’ in ERC20 and IERC20Metadata contracts. LinearCheckpointVestingWallet’s local variable ‘checkpoints’ shadows LinearCheckpointVesting’s ‘checkpoints’ function. LinearCheckpointDualBeneficiaryVestingWallet’s constructor variable ‘checkpoints’ shadows ‘checkpoints’ functions inside LinearCheckpointVesting and LinearCheckpointVestingWallet contract.

Shadowing variables can lead to errors in the logic of code.

Files: ./contracts/WombatChildToken.sol

./contracts/LinearCheckpointVestingWallet.sol

Contracts: WombatChildToken.sol, LinearCheckpointVestingWallet.sol

Constructors: WombatChildToken, LinearCheckpointVestingWallet

Recommendation: Rename the local variables that shadow another component.

Status: Fixed (Revised commit: b4091c1e5015152aad8a2573bf85b26d87f39fcf)

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.