SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Manilla Finance
Date: 18 August, 2023

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

This report may contain confidential information about IT systems and the intellectual property of the Customer, as well as information about potential vulnerabilities and methods of their exploitation.

The report can be disclosed publicly after prior consent by another Party. Any subsequent publication of this report shall be without mandatory consent.

Document

Name	Smart Contract Code Review and Security Analysis Report for Manilla Finance
Approved By	Paul Fomichov | Lead Solidity SC Auditor at Hacken OU
Tags	BEP-20 token
Platform	EVM
Language	Solidity
Methodology	Link
Website	https://website.com
Changelog	25.07.2023 - Initial Review 09.08.2023 - Second Review 14.08.2023 - Third Review 18.08.2023 - Fourth Review

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

Table of contents

Introduction **4**
System Overview **4**
Executive Summary **5**
Recommendations **6**
Checked Items **7**
Findings **10**
Critical 10
High 10
H01. Requirements Violation 10
Medium 10
M01. Data Consistency 10
Low 11
L01. Floating Pragma 11
L02. Missing Zero Address Validation 11
Disclaimers **12**
Appendix 1. Severity Definitions **13**
Risk Levels 13
Impact Levels 14
Likelihood Levels 14
Informational 14
Appendix 2. Scope **15**

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

Introduction

Hacken OÜ (Consultant) was contracted by Manilla Finance (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.

System Overview

PROJECT_NAME is a staking protocol with the following contracts:

- *Manilla* – simple ERC-20 token that mints all initial supply to a deployer. Additional minting is not allowed.
  It has the following attributes:
  - Decimals: 18
  - Total supply: 1b tokens stated in the docs.

Privileged roles

- The Owner can transfer ownership to another account.

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 scoring methodology.

Documentation quality

The total Documentation Quality score is **10** out of **10**.

- Technical description is provided.
- NatSpec is provided and extensive.

Code quality

The total Code Quality score is **10** out of **10**.

- The development environment is configured.
- The code follows the style guides and best practices.

Test coverage

Code coverage of the project is **50%** (branch coverage), with a mutation score of 77%.

- Basic functionality is covered.
- Multiple user interactions are not covered; however, this does not affect the project score as the LoC of the project is below 250.

Security score

As a result of the audit, the code contains **no** severity 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: **10.0**. The system users should acknowledge all the risks summed up in the risks section of the report.

1	2	3	4	5	6	7	8	9	10

The final score

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
25 July 2023	2	1	1	0
09 August 2023	0	1	1	0
14 August 2023	0	0	0	0

18 August 2023	0	0	0	0

Recommendations

- Instead of implementing a custom ownable structure, use the Openzeppelin's already implemented one.
- Instead of copy-pasting the dependency contracts, import them.

Checked Items

We have audited the Customers' smart contracts for commonly known and specific vulnerabilities. Here are some items considered:

Item	Description	Status	Related Issues
Default Visibility	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed	
Integer Overflow and Underflow	If unchecked math is used, all math operations should be safe from overflows and underflows.	Passed	
Outdated Compiler Version	It is recommended to use a recent version of the Solidity compiler.	Passed	
Floating Pragma	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Passed	
Unchecked Call Return Value	The return value of a message call should be checked.	Not Relevant	
Access Control & Authorization	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	The contract should not be self-destructible while it has funds belonging to users.	Not Relevant	
Check-Effect-Interaction	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed	
Assert Violation	Properly functioning code should never reach a failing assert statement.	Passed	
Deprecated Solidity Functions	Deprecated built-in functions should never be used.	Passed	
Delegatecall to Untrusted Callee	Delegatecalls should only be allowed to trusted addresses.	Not Relevant	
DoS (Denial of Service)	Execution of the code should never be blocked by a specific contract state unless required.	Passed	

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

Race Conditions	Race Conditions and Transactions Order Dependency should not be possible.	Passed	
Authorization through tx.origin	tx.origin should not be used for authorization.	Not Relevant	
Block values as a proxy for time	Block numbers should not be used for time calculations.	Not Relevant	
Signature Unique Id	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifiers should always be used. All parameters from the signature should be used in signer recovery. EIP-712 should be followed during a signer verification.	Not Relevant	
Shadowing State Variable	State variables should not be shadowed.	Passed	
Weak Sources of Randomness	Random values should never be generated from Chain Attributes or be predictable.	Not Relevant	
Incorrect Inheritance Order	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	All external calls should be performed only to trusted addresses.	Not Relevant	
Presence of Unused Variables	The code should not contain unused variables if this is not justified by design.	Passed	
EIP Standards Violation	EIP standards should not be violated.	Passed	
Assets Integrity	Funds are protected and cannot be withdrawn without proper permissions or be locked on the contract.	Passed	
User Balances Manipulation	Contract owners or any other third party should not be able to access funds belonging to users.	Passed	
Data Consistency	Smart contract data should be consistent all over the data flow.	Passed	

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

Flashloan Attack	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. Contracts shouldn't rely on values that can be changed in the same transaction.	Not Relevant	
Token Supply Manipulation	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	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.	Not Relevant	
Style Guide Violation	Style guides and best practices should be followed.	Failed	
Requirements Compliance	The code should be compliant with the requirements provided by the Customer.	Passed	
Environment Consistency	The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Failed	
Secure Oracles Usage	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	The code should be covered with unit tests. Test coverage should be sufficient, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Passed	
Stable Imports	The code should not reference draft contracts, which may be changed in the future.	Not Relevant	

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

Findings

■■■■ Critical

No critical severity issues were found.

■■■ High

H01. Requirements Violation

Impact	High
Likelihood	Medium

The implementation of the system or function does not adhere to the high-level, broad system, technical, or functional requirements.

The provided documentation states that the total token supply is 1 billion. However, the implementation has no limitation implemented to make sure this number is not exceeded.

This can lead to minting more tokens than intended.

Path: ./contracts/Manilla.sol

Recommendation: Implement limitations to make sure that the token supply does not exceed the value set in the documentation.

Found in: 5320ee1a81e39727f39eeb7bcdd31b0f800144a9

Status: Fixed (Revised Commit: 452d9dcf1b06ceeccadf93de66170cec97a3c8e7)

■■ Medium

M01. Data Consistency

Impact	High
Likelihood	Medium

The constructor mints tokens to a user-supplied address, adminAccount. This address can be different from the caller.

This can lead to minting more tokens than intended.

Path: ./contracts/Manilla.sol

Recommendation: Make sure the adminAccount is the intended account for minting.

Found in: 5320ee1a81e39727f39eeb7bcdd31b0f800144a9

Status: Mitigated. (The issued address is provided in the deployment scripts)

Low

L01. Floating Pragma

Impact	Low
Likelihood	Medium

The project uses floating pragmas ^0.8.0.

This may result in the contracts being deployed using the wrong pragma version, which is different from the one they were tested with. For example, they might be deployed using an outdated pragma version which may include bugs that affect the system negatively.

Path: ./contracts/Manilla.sol

Recommendation: Consider locking the pragma version whenever possible and avoid using a floating pragma in the final deployment. Consider known bugs (https://github.com/ethereum/solidity/releases) for the compiler version that is chosen.

Found in: 5320ee1a81e39727f39eeb7bcdd31b0f800144a9

Status: Fixed (Revised commit: 452d9dcf1b06ceeccadf93de66170cec97a3c8e7)

L02. Missing Zero Address Validation

Impact	Low
Likelihood	Medium

Address parameters are being used without checking against the possibility of 0x0. This can lead to unwanted external calls to 0x0.

Path: ./contracts/Manilla.sol: constructor(), transferOwnership()

Recommendation: Implement zero address checks.

Found in: 5320ee1a81e39727f39eeb7bcdd31b0f800144a9

Status: Fixed (Revised commit: 452d9dcf1b06ceeccadf93de66170cec97a3c8e7)

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 based on best industry practices at the time of the writing 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 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, the Consultant cannot guarantee the explicit security of the audited smart contracts.

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

Appendix 1. Severity Definitions

When auditing smart contracts Hacken is using a risk-based approach that considers the potential impact of any vulnerabilities and the likelihood of them being exploited. The matrix of impact and likelihood is a commonly used tool in risk management to help assess and prioritize risks.

The impact of a vulnerability refers to the potential harm that could result if it were to be exploited. For smart contracts, this could include the loss of funds or assets, unauthorized access or control, or reputational damage.

The likelihood of a vulnerability being exploited is determined by considering the likelihood of an attack occurring, the level of skill or resources required to exploit the vulnerability, and the presence of any mitigating controls that could reduce the likelihood of exploitation.

Risk Level	High Impact	Medium Impact	Low Impact
High Likelihood	Critical	High	Medium
Medium Likelihood	High	Medium	Low
Low Likelihood	Medium	Low	Low

Risk Levels

Critical: Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.

High: High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation.

Medium: Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.

Low: Major deviations from best practices or major Gas inefficiency. These issues won't have a significant impact on code execution, don't affect security score but can affect code quality score.

Impact Levels

High Impact: Risks that have a high impact are associated with financial losses, reputational damage, or major alterations to contract state. High impact issues typically involve invalid calculations, denial of service, token supply manipulation, and data consistency, but are not limited to those categories.

Medium Impact: Risks that have a medium impact could result in financial losses, reputational damage, or minor contract state manipulation. These risks can also be associated with undocumented behavior or violations of requirements.

Low Impact: Risks that have a low impact cannot lead to financial losses or state manipulation. These risks are typically related to unscalable functionality, contradictions, inconsistent data, or major violations of best practices.

Likelihood Levels

High Likelihood: Risks that have a high likelihood are those that are expected to occur frequently or are very likely to occur. These risks could be the result of known vulnerabilities or weaknesses in the contract, or could be the result of external factors such as attacks or exploits targeting similar contracts.

Medium Likelihood: Risks that have a medium likelihood are those that are possible but not as likely to occur as those in the high likelihood category. These risks could be the result of less severe vulnerabilities or weaknesses in the contract, or could be the result of less targeted attacks or exploits.

Low Likelihood: Risks that have a low likelihood are those that are unlikely to occur, but still possible. These risks could be the result of very specific or complex vulnerabilities or weaknesses in the contract, or could be the result of highly targeted attacks or exploits.

Informational

Informational issues are mostly connected to violations of best practices, typos in code, violations of code style, and dead or redundant code.

Informational issues are not affecting the score, but addressing them will be beneficial for the project.

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

Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

Initial review scope

Repository	https://github.com/manillatechnologies/manilla-token
Commit	5320ee1a81e39727f39eeb7bcdd31b0f800144a9
Whitepaper	-
Requirements	Link
Technical Requirements	Link
Contracts	File: ./contracts/manilla.sol SHA3: efe00d7348b7710f58aa7cace13574d2b3064366e0d82c745ef971bacf5b20b9

Second review scope

Repository	https://github.com/manillatechnologies/manilla-token
Commit	452d9dcf1b06ceeccadf93de66170cec97a3c8e7
Contracts	File: ./contracts/manilla.sol SHA3: 8dd45422a2d27503b0ad002010c9f3d5e75d12e650d2e27ac7ebe2d4

Third review scope

Repository	https://github.com/manillatechnologies/manilla-token
Commit	093e1a41932262106ece14546cd2b759aebae2a0
Contracts	File: ./contracts/manilla.sol SHA3: 30f6c00df129c172c17851e50211742aea7938989e95cb02bc26ac13