HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Myria
Date: March 22, 2023

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 Myria
Approved By	Yevheniy Bezuhlyi | SC Audits Head at Hacken OU
Type	ERC20 token
Platform	EVM
Language	Solidity
Methodology	Link
Website	https://myria.com/
Changelog	03.03.2023 - Initial Review 22.03.2023 - 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 Myria (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://gitlab.com/myriaworld/myrianet/blockchain/myria-nft-contracts/
Commit	3d847a5eff9d7e668f24abb345e58ec3f5a318fb
Whitepaper	Not provided
Functional Requirements	Not provided
Technical Requirements	Not provided
Contracts	File: ./contracts/MyriaToken.sol SHA3: 86e30d62f5caaf9777ba7fabd99fe5dc270c0fed7996045f9c16c06c2bf41942

Second review scope

Repository	https://gitlab.com/myriaworld/myrianet/blockchain/myria-nft-contracts/
Commit	149fc0c58cd259cd5faef494ca8d6da3c678455e
Whitepaper	Myria Whitepaper v5.5 SHA3: f3d6aa9867fbad05e69e459d1b69489a6fb2332026b42ee0c3332c59d0113385
Functional Requirements	Readme file SHA3: 3d7173316f37efe68a60ac1648bcb2717bec49552e894dc27cef738055ce63d5
Technical Requirements	Readme file SHA3: 3d7173316f37efe68a60ac1648bcb2717bec49552e894dc27cef738055ce63d5
Contracts	File: ./contracts/MyriaToken.sol SHA3: bc091ac547247944c71514364f34c748771122032af4b0d669702a7b4b8cbc49

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 the loss of user funds or contract state manipulation by external or internal actors.
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 by external or internal actors.
Medium	Medium vulnerabilities are usually limited to state manipulations but cannot lead to asset loss. Major deviations from best practices are also in this category.
Low	Low vulnerabilities are related to outdated and unused code or minor Gas optimization. These issues won't have a significant impact on code execution but affect code quality

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**.

- Functional requirements, technical description and tokenomics are provided.
- The code is covered with the NatSpec comments.

Code quality

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

- The code follows the Solidity style guides.

Test coverage

Code coverage of the project is **0%** (branch coverage).

- Tests are not provided.

Security score

As a result of the audit, the code contains **1** low severity issue. 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**. The system users should acknowledge all the risks summed up in the risks section of the report.

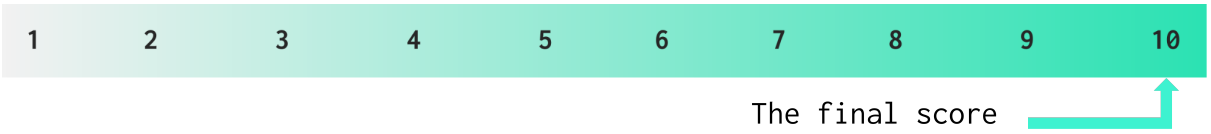

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
3 March 2023	3	3	1	0
22 March 2023	1	0	0	0

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

System Overview

- *MyriaToken* – is an ERC-20 token contract.
  It has the following attributes:
  - Decimals: 18
  - Maximal total supply: 50000000000 tokens.

  The name and the symbol are defined when the contract deployment.

  The token is ownable and allows the owner to mint tokens within the maximal total supply. The users may burn their own tokens.

Privileged roles

- The owner of the *MyriaToken* contract can mint tokens.

Risks

- The token implementation does not contain the token supply distribution, the release schedules and locks described in the tokenomics.

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

Checked Items

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

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

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 a proxy for time	SWC-116	Block numbers should not be used for time calculations.	Passed
Signature Unique Id	SWC-117 SWC-121 SWC-122 EIP-155 EIP-712	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	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 or be locked on the contract.	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

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

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 Consistency	**Custom**	The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed
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 sufficient, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Failed
Stable Imports	**Custom**	The code should not reference draft contracts, which may be changed in the future.	Passed

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. Weak Auth System

The contract functionality allows to add the minted managers using the *setMintManagers* function, but not to remove them.

Such role-granting functionality is unsafe as it can lead to manipulations in case the manager was added incorrectly or was compromised.

Path: ./contracts/MyriaToken.sol : setMintManagers()

Recommendation: add the functionality for the minted managers' removal.

Found in: 3d847a5

Status: Fixed. Functionality removed (Revised commit: 149fc0c)

Medium

M01. Requirements Violation

The *mint* function has the amount parameter and no *account* parameter as it is required in the standard mintable extension.

Such behavior is not documented and may indicate that the requirements are violated. This may lead to incorrect function usage.

Path: ./contracts/MyriaToken.sol : mint()

Recommendation: clarify the minting functionality, document it, and ensure the implementation matches the requirements.

Found in: 3d847a5

Status: Fixed. NatSpec added (Revised commit: 149fc0c)

M02. Best Practice Violation

The *lockPeriods* parameter's length is not checked for equality with the *approvedMinters* and *allowances* parameters' lengths.

This may lead to incorrect input data processing.

Path: ./contracts/MyriaToken.sol : setApprovedMinters()

Recommendation: verify if the *lockPeriods* parameter's length is equal to the *approvedMinters* and *allowances* parameters' lengths.

Found in: 3d847a5

Status: Fixed. Functionality removed (Revised commit: 149fc0c)

M03. Missing Event Emitting

Critical state changes should emit events for tracking things off-chain.

The following functions do not emit events on the change of important values.

Path: ./contracts/MyriaToken.sol : setMintManagers(), setApprovedMinters()

Recommendation: create and emit the event whenever the corresponding action happens.

Found in: 3d847a5

Status: Fixed. Functionality removed (Revised commit: 149fc0c)

Low

L01. Best Practice Violation

The contract uses the *_msgSender* function from the OpenZeppelin *Context* contract for the most functionality. However, the *msg.sender* is used in the *burn* function.

Using different styles for the *msg.sender* obtaining decreases the code readability.

Path: ./contracts/MyriaToken.sol : burn()

Recommendation: obtain the *msg.sender* in the same way in all the contract functionality.

Found in: 3d847a5

Status: Fixed (Revised commit: 149fc0c)

L02. Usage Of Block Values For Time Calculations

The contracts use *block.timestamp* for time calculations.

It is not precise and safe.

Path: ./contracts/MyriaToken.sol : setApprovedMinters(), mint()

Recommendation: avoid using the block values for the time calculations. Alternatively, it is safe to use oracles.

Found in: 3d847a5

Status: Fixed. Functionality removed (Revised commit: 149fc0c)

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

L03. Redundant Variable Usage

The *DECIMALS* variable is redundant as the *decimals()* function from the inherited OpenZeppelin *ERC20* contract can be used for the required calculation.

Redundant code decreases the code readability.

Path: ./contracts/MyriaToken.sol : DECIMALS

Recommendation: replace the *DECIMALS* variable usage with the *decimals()* function.

Found in: 3d847a5

Status: Fixed (Revised commit: 149fc0c)

L04. Best Practice Violation

There is *uint* value in the contract whose size is not set explicitly.

This decreases the code readability. The explicit type definition helps to understand the data size to proceed and detect errors.

Path: ./contracts/MyriaToken.sol : burn() - amount parameter

Recommendation: replace the *uint* type with the *uint256*.

Found in: 149fc0c

Status: New

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.