HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Muon
Date: 09 Aug, 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 Muon
Approved By	Oleksii Zaiats | Head of Solidity SC Auditor department at Hacken OU
Type	ERC20 token; ERC721 token;
Platform	EVM
Language	Solidity
Methodology	Link
Website	https://www.muon.net/
Changelog	17.07.2023 - Initial Review 09.08.2023 - Second Review

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

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by Muon (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

Token - an upgradable, pausable, and burnable ERC20 token smart contract, with specific roles for pausing the contract and minting new tokens.

BondedToken - a custom implementation of an ERC721 (non-fungible token) smart contract that provides functionalities for minting, burning, and transferring tokens. Additionally, it has unique features such as token whitelisting, locking of assets in tokens, as well as splitting and merging tokens while maintaining locked assets.

PION - an ERC20 token based on *Token*. It has the following attributes:

- Name: PioneerNetwork
- Symbol: PION
- Decimals: 18
- Total supply: unlimited

BondedPION - an ERC721 token based on *BondedToken*.

IToken - an interface for ERC20 tokens.

Privileged roles

- Token.sol:
  - Admin:
    - Manage other roles throughout users.
  - Pauser:
    - Pause and unpause smart contract.
  - Minter:
    - Mint unlimited amount of tokens.
- BondedToken.sol:
  - Owner:
    - Pause and unpause smart contract. Set tokens whitelist, enable/disable public transfers, set treasury address.
  - Admin:
    - Manage other roles throughout users.
  - Transferable address:
    - Users with only this role can receive and send tokens.

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 **7** out of **10**.

- Functional requirements have some gaps:
  - No roles description.
  - Project overview is detailed
  - All interactions are described.
- Technical description is inadequate:
  - Run instructions are provided.
  - Technical specification is not provided.
  - NatSpec is sufficient.

Code quality

The total Code Quality score is **9** out of **10**.

- Best practice violations.

Test coverage

Code coverage of the project is **92.86%** (branch coverage).

- Tests are provided.
- Negative cases coverage is present.
- Interactions by several users are tested.

Security score

As a result of the audit, the code contains **no** 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: **9.2**. The system users should acknowledge all the risks summed up in the risks section of the report.

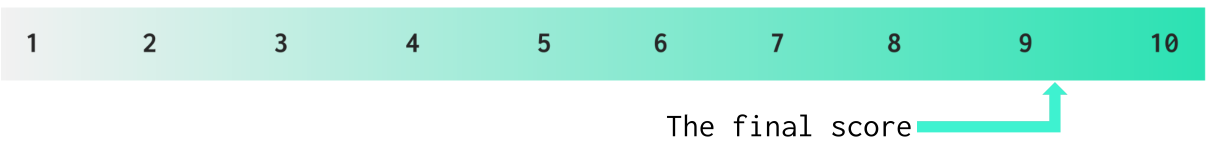

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
17 July 2023	1	4	0	0
09 August 2023	0	0	0	0

Risks

No potential security risks were found during the audit research.

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	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	

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.	Passed	
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.	Not Relevant	
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	

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.	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.	Passed	
Style Guide Violation	Style guides and best practices should be followed.	Failed	I01, I03, I08
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.	Passed	
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.	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

No high severity issues were found.

Medium

M01. Denial of Service; Inefficient Gas Modeling

Impact	Medium
Likelihood	Medium

The *lock()* function gets a parameter list of tokens to be locked. If one token in a list was not whitelisted, the whole transaction would be reverted.

This leads to the fact that due to one error, the entire transaction will be reverted, but the fees were paid.

Paths:
./contracts/BondedToken.sol : lock();

Recommendation: Consider skipping the loop for tokens that are not in the whitelist.

Found in: 78d591d

Status: Fixed (Revised commit: 22476a2).

M02. Coarse-Grained Authorization Model; Inefficient Gas Modeling;

Impact	Medium
Likelihood	Medium

Bonded token smart contract uses OpenZeppelin AccessControl and Ownable. In the *_initialize()* function, the admin role and owner variable are set to the same address(*msg.sender*).

Several functions use *onlyOwner* (Ownable) modifier, but they can be used with *onlyRole(DEFAULT_ADMIN_ROLE)* (AccessControl) instead.

This overuse of Ownable leads to inefficient Gas usage and possible errors when working with a smart contract.

Paths:
./contracts/BondedToken.sol : pause(), unpause(), whitelistTokens(), setPublicTransfer(), setTreasury();

Recommendation: Remove Ownable usage, change modifiers for relevant functions.

Found in: 78d591d

Status: Mitigated (with Customer notice: *We do not want to have more than one owner. With AccessControl, the role could be assigned to more than one wallet. Some of the NFT platforms allow the owner of an ownable token to customize some settings.And of course, msg.sender is the default owner and will be updated later.*)

M03. Missing Storage Gaps

Impact	Medium
Likelihood	Medium

When working with upgradeable contracts using OpenZeppelin Upgrades, it is necessary to introduce storage gaps to make the project compatible.

Storage gaps are a convention for reserving storage slots in a base contract, allowing future versions of that contract to use up those slots without affecting the storage layout of child contracts.

Paths:
./contracts/BondedToken.sol
./contracts/Token.sol

Recommendation: Introduce Storage Gaps in the contract.

To create a storage gap, declare a fixed-size array in the base contract with an initial number of slots. This can be an array of *uint256* so that each element reserves a 32 byte slot. Use the name *__gap* or a name starting with *__gap_* for the array so that OpenZeppelin Upgrades will recognize the gap.

Found in: 78d591d

Status: Mitigated (with Customer notice: *We do not need to change the types of any fields and there are no Struct fields that need __gap. We will add new fields at the end of the existing fields if it is necessary to add any new fields in the upgrades.*)

M04. CEI Pattern Violation

Impact	Medium
Likelihood	Medium

The Checks-Effects-Interactions pattern is violated. During the *lock()* function, state variables are updated after the external calls.

Path:
./contracts/BondedToken.sol : lock();

Recommendation: Implement the function according to the Checks-Effects-Interactions pattern.

Found in: 78d591d

Status: Mitigated (with Customer notice: *Following an in-depth discussion with the client, we concluded that this issue does not result in security problems; rather, it pertains more to best practices.*)

Low

L01. Missing Events

Impact	Low
Likelihood	Medium

Events should be emitted after sensitive changes take place, to facilitate tracking and notify off-chain clients following the contract's activity.

Path:
./contracts/BondedToken.sol : mint(), burn();

Recommendation: Consider emitting events in said functions.

Found in: 78d591d

Status: Fixed (Revised commit: 22476a2).

Informational

I01. Functions That Should Be External

Public functions that are not called from inside the contract should be declared external to save Gas.

Paths:
./contracts/Token.sol : pause(), unpause(), mint();
./contracts/PION.sol : initialize();
./contracts/BondedPION.sol : initialize();
./contracts/BondedToken.sol : pause(), unpause(), burn(), supportsInterface();

Recommendation: Consider changing the function visibility to external.

Found in: 78d591d

Status: Reported (Revised commit: 22476a2 : Functions that should be external ./contracts/BondedToken.sol : burn(), supportsInterface()).

I02. Solidity Style Guide Violation: Order Of Layout

Inside each contract, library or interface, use the following order:

1. Type declarations
2. State variables
3. Events
4. Errors
5. Modifiers
6. Functions
    a. constructor
    b. initializer (if exists)
    c. receive function (if exists)
    d. fallback function (if exists)
    e. external
    f. public
    g. internal
    h. private

Paths:

./contracts/BondedToken.sol;
./contracts/Token.sol;

Recommendation: Change order of layout to fit Official Style Guide.

Found in: 78d591d

Status: Fixed (Revised commit: 22476a2).

I03. Redundant Function Virtualization

The following functions are marked as virtual in the code, but never being overridden. Virtual functions are much more Gas expensive compared to default functions.

Path:
./contracts/BondedToken.sol : burn(), supportsInterface();

Recommendation: Make these functions non-virtual.

Found in: 78d591d

Status: Reported

I04. Boolean Equality

Boolean constants can be used directly and do not need to be compared to true or false.

Path:
./contracts/BondedToken.sol : whitelistTokens();

Recommendation: Remove boolean equality.

Found in: 78d591d

Status: Fixed (Revised commit: 22476a2).

I05. Solidity Style Guide Violation: mixedCase in State Variables Names

Local and State Variable names should be mixedCase: capitalize all the letters of the initialisms, except keep the first one lower case if it is the beginning of the name.

Path:

./contracts/BondedToken.sol;

Recommendation: follow the official Solidity guidelines.

Found in: 78d591d

Status: Fixed (Revised commit: 22476a2).

I06. Inefficient Gas Modeling

Inside loop the value of *list.length* taken every loop iteration. Declaring a variable equal to the length of the list before the loop will reduce the Gas consumption when deploying a smart contract.

Path:

./contracts/BondedToken.sol : whitelistTokens(), burn(), merge();

Recommendation: Declare variable above loop to decrease.

Found in: 78d591d

Status: Fixed (Revised commit: 22476a2).

I07. Unnecessary Value Assignment

The value of the *isPublicTransferEnabled* state variable is *false* by default. Setting the value of a variable is redundant.

Path:
./contracts/BondedToken.sol : _initialize()

Recommendation: Remove value assignment.

Found in: 78d591d

Status: Fixed (Revised commit: 22476a2).

I08. Missing Event Indexes

Use indexed events to keep track of a smart contract's activity after it is deployed, which is helpful in reducing overall Gas.

Path:
./contracts/BondedToken.sol : WhitelistTokensUpdated, PublicTransferStatusUpdated, TreasuryUpdated;

Recommendation: Add indexed keyword.

Found in: 78d591d

Status: Reported

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.

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/muon-protocol/muon-tokenomics-contracts
Commit	78d591d
Whitepaper	https://rtd-muon.readthedocs.io/en/latest/introduction.html
Requirements	https://github.com/muon-protocol/muon-tokenomics-contracts/wiki
Technical Requirements	https://github.com/muon-protocol/muon-tokenomics-contracts/blob/main/README.md
Contracts	File: ./contracts/BondedPION.sol SHA3: 6e83cb922c0aeffbf850cdc69fd4da8b62baf945bcb1d5f186ec1fc67914ae6a File: ./contracts/BondedToken.sol SHA3: 1a1229833523731e1c54cf6996ab0284dc7681f13eb59a7ed8215ea28931ec76 File: ./contracts/PION.sol SHA3: 62cdab091606a5f39a23d9e8a33d44c45b891a58852eb1f0cd1a629c5af026e1 File: ./contracts/Token.sol SHA3: 68ac39c24f72266f0dff465afe42b69859b798971135fbd4c1c3313bf27bfc62 File: ./contracts/interfaces/IToken.sol SHA3: 66d7862c2bd6d02824b7ed79c2204fe1e34d08a3bf2b9920a9ca82dce18b5940

Second review scope

Repository	https://github.com/muon-protocol/muon-tokenomics-contracts
Commit	22476a2
Whitepaper	https://rtd-muon.readthedocs.io/en/latest/introduction.html
Requirements	https://github.com/muon-protocol/muon-tokenomics-contracts/wiki
Technical Requirements	https://github.com/muon-protocol/muon-tokenomics-contracts/blob/main/README.md
Contracts	File: ./contracts/PION.sol SHA3: 4b8f6d2f29046fab64306c4044b4e1ec292da6ab8db6682f299952a9331fd9a3 File: ./contracts/Token.sol SHA3: 2a7a3b016e18a7d788d337d8e9aee837461599464a2f6e9c1818a6aab20d6bf3 File: ./contracts/BondedPION.sol SHA3: 2f7f4d8de2dfab763b5050c720dd182b54be9186762536e9d95582d4760f0ba3

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

	File: ./contracts/BondedToken.sol SHA3: 948ae2a2946d5c8e42195556c17ab19663d0f2f2b4bdef8f80207ba661662090 File: ./contracts/interfaces/IToken.sol SHA3: 66d7862c2bd6d02824b7ed79c2204fe1e34d08a3bf2b9920a9ca82dce18b5940