HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Qoodo.io
Date: May 24, 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 Qoodo.io
Approved By	Paul Fomichov | Lead Smart Contract Auditor at Hacken OU
Type	ERC20 token; Staking
Platform	EVM
Language	Solidity
Methodology	Link
Website	Qoodo.io
Changelog	13.04.2023 - Initial Review 24.05.2023 - 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 Qoodo.io (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 includes the following smart contracts from the provided repository:

Initial review scope

Repository	https://github.com/NAPLOZZ/NAP-token-contracts
Commit	981b71e
Functional Requirements	http://docs.naplozz.io/
Technical Requirements	NAP TOKEN DOC.pdf SHA3: 31c465b7043d68b9a3ce7a2cd8aeebc220799ac01aacf9bd1d3f4cafd778c14c
Contracts	File: ./contracts/Ecosystempool.sol SHA3: 8ec11193c561562cce7e64ccd6142ca9ca9f8a5e8680d655556f93a5c51cce21 File: ./contracts/NaplozzToken.sol SHA3: 1c9783b1b94c776e8fc2b494fa784206e930918a7dd02773f13ec07e14930082 File: ./contracts/StakingNapTokens.sol SHA3: fd0c276b5c25163f5a160ca59d9aa3f98b799fd2d0ba34559782443a1302110b File: ./contracts/TokenDistributor.sol SHA3: e644e3f2fd3b4d80e273d18c4f80e6e0ec47ca9d92b44a1a9a9cc54d2396c75c File: ./contracts/TokenSplitter.sol SHA3: 3d78b0837c6cfd3375bab3148d2dd1a04db63b2cb7dd980441691c93c03a042b File: ./contracts/Yearlyvesting.sol SHA3: 51411fe307ba004b4d305f089934865b30ca1ed08c116a1d365783efe015c8f3 File: ./contracts/interfaces/ILooksRareToken.sol SHA3: 46ef3cc1ae10f4ae4d0822023844e7e96c469729b539149a82b85cc1dd8c8a67

Second review scope

Repository	https://github.com/NAPLOZZ/NAP-token-contracts
Commit	6bee1f4
Functional Requirements	http://docs.naplozz.io/
Technical Requirements	NAP TOKEN DOC.pdf SHA3: 31c465b7043d68b9a3ce7a2cd8aeebc220799ac01aacf9bd1d3f4cafd778c14c

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

Contracts	File: ./contracts/EcoSystemPool.sol SHA3: 67362788138084fb77177336c3986be99cd5b0f392823da4e628456886a26687 File: ./contracts/NaplozzToken.sol SHA3: b166017ee3c09ce689a36d1cd0fb582a46419db4632b6226bd0fc2535ee529e8 File: ./contracts/StakingNapTokens.sol SHA3: a6505bde3e878a3c92c5c713e78556af57687b5db2f74cb897838bb9afffa817 File: ./contracts/YearlyVesting.sol SHA3: d7f704809cc2b19dce5942ab4bc93a1a272997dcb9ed516cb72421fa2708d455

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 **9** out of **10**.

- Functional requirements are partially missed.
- Technical description is partially missed.

Code quality

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

- The development environment is configured.
- The project follows Solidity style guides.

Test coverage

Code coverage of the project is **93%** (branch coverage).

Security score

As a result of the audit, the code contains **0** 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: **9.7**. The system users should acknowledge all the risks summed up in the risks section of the report.

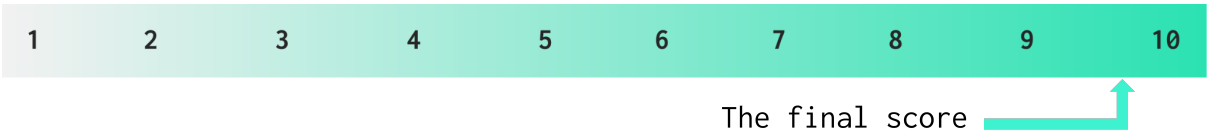

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
13 April 2023	12	0	5	2
24 May 2023	0	0	0	0

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

Risks

- The contract NaplozzToken interacts with *antisnipe* contract in *_beforeTokenTransfer()* function, but its source cannot be verified for safety.

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

System Overview

Qoodo.io is a mixed-purpose system with the following contracts:

- *NaplozzToken* – ERC-20 token.
  It has the following attributes:
  - Name: NAPLOZZ
  - Symbol: NAP
  - Decimals: 18
  - Total supply: no cap.
- *StakingNapTokens* – a contract that rewards users for staking their NAP tokens. The reward can be set by the contract owner.
- *TokenDistributor* – a contract that handles the distribution of ERC20 tokens. The contract distributes tokens over a set number of periods, and the reward per block adjusts automatically over the set periods.
- TokenSplitter – a contract that splits tokens among a group of accounts based on predefined shares.
- YearlyVesting – a vesting contract. Each contract handles the vesting process for a single user.
- EcoSystemPool – documentation not provided, unclear functionalities.

Privileged roles

- The owner of the *TokenSplitter* contract can arbitrarily update user shares.
- The owner of *StakingNAPTokens* can pause and unpause the contract, withdraw the rewards balance, and update the staking reward rate.
- The *NaplozzToken* contract role *MINTER_ROLE* can arbitrarily mint new tokens.
- The *NaplozzToken* contract role *BURNER_ROLE* can arbitrarily burn tokens from any user.
- The owner of the *EcoSystemPool* contract can update the token address.

Recommendations

- In Ecosystempool the function name init() is misleading as it is a normal setter and not an initialization function. It is better to properly give names to functions and variables in order to be intuitive and describe their functions.
- The withdraw() functions of TokenDistributor and StakingNapToken have a typo in the error message, ‘or’ instead of ‘and’. It is better to describe the error as it is in order to provide the best user experience to the user that receives the error.

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.	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.	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.	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.	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.	Passed
Stable Imports	**Custom**	The code should not reference draft contracts, which may be changed in the future.	Passed

Findings

■■■■ Critical

C01. Access Control Violation

The function *transfer()* allows any caller to withdraw all the contract funds.

The input parameter *addr* is only used in the emitted event, the tokens are sent to the msg.sender in any case. But even if *addr* was properly used, the critical issue would persist.

Path: ./contracts/Ecosystempool.sol : transfer()

Recommendation: From the provided documentation, it seems this function was meant to implement the *onlyOwner* modifier, but more context from the client would be useful.

Found in: 981b71e

Status: Fixed (Revised commit: 7594909)

C02. Highly Permissive Role Access

The *adminRewardWithdraw()* function allows the contract owner to withdraw the reward tokens belonging to the contract.

This would be an issue by itself, as there is no time-lock or check on the staked funds. The real issue is that according to the documentation, the staking token and the reward token are both the NaplozzToken, allowing the contract owner to withdraw the users staked funds.

Path: ./contracts/StakingNapTokens.sol : adminRewardWithdraw()

Recommendation: A possible solution is to just remove the function. Another solution would be to use a different staking token, and only allow the withdrawal of the portion of rewards not belonging to ongoing stakings.

Found in: 981b71e

Status: Fixed (Revised commit: 7594909)

■■■ High

H01. Race Condition

The *lock()* function allows a front-running attacker to transfer funds from users that approve the contract to spend their tokens before the user actually performs the locking.

Path: ./contracts/Yearlyvesting.sol : lock()

Recommendation: Assure that the *_from* parameter and *msg.sender* are the same, or that *msg.sender* is an authorized user.

Found in: 981b71e

Status: Fixed (Revised commit: 7594909)

H02. Highly Permissive Role Access

The function *burn()* allows *BURNER_ROLE* to burn tokens from arbitrary addresses.

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

Recommendation: Change the *burn()* function to explicitly require the approval of a user to burn their tokens or, as usual, only allow users to burn their own tokens. If that is really intentional, it should be highlighted in the documentation.

Found in: 981b71e

Status: Fixed (Revised commit: 7594909)

H04. Insufficient Balance

The contract does not implement any kind of accountancy of the rewards. Users cannot be assured they will receive the staking rewards.

The issue is made worse by the fact that the staking token and the reward token are both the NaplozzToken, opening for the possibility of user staked balances being used for other users' rewards.

Path: ./contracts/StakingNapTokens.sol

Recommendation: Implement rewards balance accountancy - while doing so, pay attention to the *updateRewardPerBlockAndEndBlock()* function and the *endBlock* variable being written, as it can lead to math issues.

Found in: 981b71e

Status: Fixed (Revised commit: 7594909)

H06. Insufficient Balance

In the *deposit()* function the reward token balance is not properly checked. It is currently checked that *rewardTokenBalance > 0*, but this check does not take into account accrued but unwithdrawn rewards.

Path: ./contracts/StakingNapTokens.sol : deposit()

Recommendation: The check to be enforced is *rewardTokenBalance* > *accruedRewards*, where *accruedRewards* is the product of *rewardPerBlock* and the blocks elapsed since *START_BLOCK*.

Found in: 7594909

Status: Fixed (Revised commit: 6bee1f4)

Medium

No medium severity issues were found.

Low

L01. Floating Pragma

Locking the pragma helps ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively. The project uses floating pragmas >=0.8.0 < 0.8.4.

Paths: ./contracts/Ecosystempool.sol

./contracts/NaplozzToken.sol

./contracts/StakingNapTokens.sol

./contracts/Yearlyvestin.sol

Recommendation: Consider locking the pragma version whenever possible and avoid using a floating pragma in the final deployment.

Found in: 981b71e

Status: Fixed (Revised commit: 7594909)

L02. Redundant Variable

locked variable is redundant as the same functionality can be achieved by checking *expiry* variable, saving Gas on operations and deployment.

Path: ./contracts/Yearlyvesting.sol

Recommendation: Remove redundant variables.

Found in: 981b71e

Status: Fixed (Revised commit: 7594909)

L03. Redundant Cast

Address variables often get unnecessarily cast to address type.

Paths: ./contracts/Yearlyvesting.sol : lock(), withdraw()

./contracts/Ecosystempool.sol : transfer()

Recommendation: Remove redundant castings.

Found in: 981b71e

Status: Fixed (Revised commit: 7594909)

L04. Redundant Require

In multiple functions, *msg.sender* is checked to be != address(0), but this can never happen.

Paths: ./contracts/Yearlyvesting.sol : withdraw()

./contracts/Ecosystempool.sol : transfer()

Recommendation: Remove redundant require statement.

Found in: 981b71e

Status: Fixed (Revised commit: 7594909)

L05. Unfinalized Code

Some contract files contain commented code parts or useless comments, meaning that the code is not finished yet.

Paths: ./contracts/Ecosystempool.sol : line 4

./contracts/Yearlyvesting.sol : lines 34, 51

Recommendation: Removed commented code parts and useless comments.

Found in: 981b71e

Status: Fixed (Revised commit: 7594909)

L06. 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.

Paths: ./contracts/Ecosystempool.sol : init()

./contracts/NaplozzToken.sol : setAntisnipeAddress()

Recommendation: Implement zero address checks.

Found in: 981b71e

Status: Fixed (Revised commit: 7594909)

L07. Missing Error Message

The *require* statement in setAntisnipeDisable() does not include an error message.

Path: ./contracts/NaplozzToken.sol : setAntisnipeDisable()

Recommendation: Add an error message to improve the user experience.

Found in: 981b71e

Status: Fixed (Revised commit: 7594909)

L08. Lack of Event Emission

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

Path: ./contracts/StakingNapTokens.sol : deposit(), withdraw()

Recommendation: Emit harvest event when it happens in deposit() and withdraw() functions.

Found in: 981b71e

Status: Fixed (Revised commit: 7594909)

L09. Inefficient Gas Model

In the function _updatePool() the check at line 318 can be moved before the computation of *multiplier*, since in the case of *lastRewardBlock* >= *endBlock* *multiplier* will be 0 and the function can return. This will save Gas on computations.

Path: ./contracts/StakingNapTokens.sol

Recommendation: Before the computation of multiplier, return if lastRewardBlock >= endBlock.

Found in: 981b71e

Status: Fixed (Revised commit: 7594909)

L10. Bad Function Naming

The contract EcoSystemPool function init() does not initialize anything, only updates the value of the variable *token*.

Path: ./contracts/Ecosystempool.sol

Recommendation: Change the function name in order to be intuitive.

Found in: 981b71e

Status: Fixed (Revised commit: 7594909)

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

L11. Style Guide Violation

The provided projects should follow the official guidelines.

Path: all contracts

Recommendation: Follow the official Solidity guidelines.

Found in: 981b71e

Status: Fixed (Revised commit: 7594909)

L12. Functions that Can Be Declared External

public functions that are never called by the contract should be declared *external* to save Gas.

Path: ./contracts/NaplozzToken.sol : burn(), mint()

Recommendation: Change functions visibility to *external*.

Found in: 981b71e

Status: Fixed (Revised commit: 7594909)

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.