SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: YellowNetwork_Layer-3 Foundation
Date: 2 March, 2023

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 YellowNetwork Layer-3 Foundation
Approved By	Yevheniy Bezuhlyi | SC Audits Head at Hacken OU
Type	ERC20 token
Platform	EVM
Language	Solidity
Methodology	Link
Changelog	22.02.2023 - Initial Review 02.03.2023 - Second Review

Table of contents

Introduction **4**

Scope **4**

Severity Definitions **5**

Executive Summary **6**

System Overview **7**

Checked Items **8**

Findings **11**
- Critical 11
- High 11
  - H01. Highly Permissive Role Access 11
- Medium 11
- Low 11
  - L01. Redundant view functions 11

Disclaimers **12**

Introduction

Hacken OÜ (Consultant) was contracted by YellowNetwork Layer-3 Foundation (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/layer-3/clearsync
Commit	9ab5313ac5b3b7ed9ba402de86340394b88c7bd1
Whitepaper	Not provided
Functional Requirements	Link
Technical Requirements	Link
Contracts	File: ./contracts/Token.sol SHA3:dd87ed35778bd10e5b55f5ccf37ff1a37ceaf0551a1ad78268c15e187acca440 File: .contracts/interfaces/IBlacklist.sol SHA3 73757b8881c1d29a703eaeec30e81f23bbb386b42a6f5595a9c16a8556e748f1

Second review scope

Repository	https://github.com/layer-3/clearsync
Commit	5b86a2134d295ac11af97d4f239782222e95fe24
Whitepaper	Link
Functional Requirements	Link
Technical Requirements	Link
Contracts	File: ./contracts/Token.sol SHA3:9b995473862b428b32a5cab75e4c8aae2903c1c8b9cc6ef29fc4bd35057c2d5f File: .contracts/interfaces/IBlacklist.sol SHA3:76a94f93ff4eddecd5cf6ac732a573ab3435729fc43a48eee87cff2806b11621

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

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**.

- Business logic is provided.
- Use cases are provided.
- Whitepaper is provided.
- Functional requirements are provided.

Code quality

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

- NatSpec is present.
- Code follows Solidity style guidelines.

Test coverage

Code coverage of the project is **95.83%** (branch coverage).

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**.

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
22 Feb 2023	0	0	1	0
2 Mar 2023	1	0	0	0

System Overview

YellowNetwork is a simple system with the following contracts:

- Token – simple ERC-20 token that allows mint, transfer and burn tokens with addition of blacklist. Addresses on blacklist cannot transfer tokens. While deploying contract developer needs to provide following attributes:
    - Name: token name
    - Symbol: token symbol
    - Supply Cap: minting over cap is not allowed.

  Contract uses roles to restrict access to important functions.

- IBlacklist – an interface that contains functions and events for blacklisting mechanisms.

Privileged roles

- DEFAULT_ADMIN_ROLE - address with that role can activate minting tokens.
- COMPLIANCE_ROLE - address with that role can add and remove given address from blacklist. Additionally, that role allows to burn all tokens from blacklisted addresses.
- MINTER_ROLE - address with that role can mint new tokens.

Risks

- **An account with the COMPLIANCE_ROLE can blacklist and then an account with DEFAULT_ADMIN_ROLE can burn tokens of any account without allowance. Tokens can be burned from contracts such as LPs.**
- The repository contains contracts that are out of the audit scope. Secureness and reliability of those contracts may not be guaranteed by the current audit.

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

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.	Not Relevant
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.	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.	Not Relevant
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

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.	Not Relevant
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

No critical severity issues were found.

■■■ High

H01. Highly Permissive Role Access

An account with the COMPLIANCE_ROLE can blacklist and then burn tokens of any account without allowance. Tokens can be burned from contracts such as LPs, which can lead to market manipulation.

Path: ./contracts/Token.sol: burnBlacklisted()

Recommendation: Do not burn tokens without allowance.

Status: Mitigated (The functionality is needed to protect the protocol. COMPLIANCE_ROLE can blacklist accounts. DEFAULT_ADMIN_ROLE can burn blacklisted funds. Both roles should be granted only to multisigs with ⅔ signatures required.)

■■ Medium

No medium severity issues were found.

■ Low

L01. Redundant View Functions

The variable *TOKEN_SUPPLY_CAP* is marked public. View function for the public variable is generated automatically by the compiler.

Path: ./contracts/Token.sol: cap()

Recommendation: Delete *cap()* function or mark *OKEN_SUPPLY_CAP* variable as private.

Status: New

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.