HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: ReduX Technologies AG
Date: November 25th, 2022

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 ReduX Technologies AG
Approved By	Evgeniy Bezuglyi | SC Audits Department Head at Hacken OU
Type	BEP20 token; Market
Platform	EVM
Language	Solidity
Methodology	[Link](#)
Website	
Changelog	18.11.2022 – Initial Review 25.11.2022 – Second Review

Table of contents

www.hacken.io

Introduction

Hacken OÜ (Consultant) was contracted by ReduX Technologies AG (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/goreduxnow/token
Commit	f58de559a1d001c91e9ca6f32c77dd8e7048037f
Repository	https://github.com/goreduxnow/ICO
Commit	f5e12e1fe432efa2b546cd037c4d86a929744485
Whitepaper	Link
Functional Requirements	Link
Contracts	File: ./contracts/ReduxToken.sol SHA3: b4e81bfd55faf552bdefdea46b43e2ae8458049345d1c291afc4893996752cb8 File: ./contracts/OwnerWithdrawable.sol SHA3: bcc00ee11b4a4362b1dedd3dd1bfe831fb9dd2815cabf154f996739b513b0a72 File: ./contracts/preSale.sol SHA3: fbdb02ceed364001abbaaa0cda8cd78cc9f4a92611ed21b3a231e3521d3415af

Second review scope

Repository	https://github.com/goreduxnow/token
Commit	a893ff608584cb27a7d8a66c4338ac0992eebe20
Repository	https://github.com/goreduxnow/ICO
Commit	31fc9932e7ccde4ec60b9fbb270610b5cbd7ebc2
Whitepaper	Link
Functional Requirements	ICO, Token
Contracts	File: ./contracts/ReduxToken.sol SHA3: 91f3e1e1fabc06bceaa02e43d85df88cb5c3cb9291d706769a079fbc23d316b2 File: ./contracts/OwnerWithdrawable.sol SHA3: 109677546d8e03b27b1b5914f3446b82d8398344d5833fef14f890cde4cd443a File: ./contracts/preSale.sol SHA3: 67da67589d95a06bf017dea74259c5cfecbd77a5c3dbcccd4f87b65779b5d4fb

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 assets 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 the 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 **9** out of **10**.
- Functional requirements are superficial.
- Code is followed by NatSpec comments.

Code quality

The total Code Quality score is **8** out of **10**.
- Code violates Style guide.

Test coverage

Test coverage of the project is **100%** (branch coverage).
- Project`s total lines of code are less than 250.

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: **9.5**.
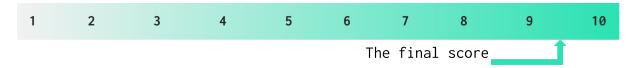
1	2	3	4	5	6	7	8	9	10

The final score ⬆

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
17 November 2022	5	3	1	1
24 November 2022	0	0	0	0

www.hacken.io

Checked Items

We have audited the Customers' smart contracts for commonly known and more 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.	Passed
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	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	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.	Passed
Assets integrity	Custom	Funds are protected and cannot be withdrawn without proper permissions.	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.	Passed
Style guide violation	**Custom**	Style guides and best practices should be followed.	Failed
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 100%, 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

www.hacken.io

System Overview

ReduX is a BEP20 token and a sales market with following contracts:
- *Redux* — simple BEP-20 token that mints all initial supply to a deployer. Additional minting is not allowed.
  It has the following attributes:
    - Name: ReduX
    - Symbol: REDUX
    - Decimals: 18
    - Total supply: $2*10^{27}$ tokens.
- *OwnerWithdrawable* — a contract with the functionality to withdraw balance to the owner's account.
- preSale - a marketplace for the ReduX token.

Privileged roles

- The owner of *ReduX* token receives all the initial mint amount and can burn tokens, pause and unpause the contract.
- The owner of *OwnerWithdrawable* can withdraw the contract`s balance.
- The owner of *preSale* contract can change sale token parameters, set sale period, set vesting period, and set external allocation.

Findings

■■■■ Critical

1. Funds Lock

Native coins and tokens should have mechanisms of withdrawal from the contract if they are accepted by the contract. Such functionality is not provided, thus permanently locking all the unsold Redux tokens from circulation.

Path: ./contracts/preSale.sol

Recommendation: Implement a function to withdraw the unsold tokens after the sale ends.

Status: Fixed (revised commits: a893ff6, 31fc993)

■■■ High

2. Checks-Effects-Interactions Pattern Violation

The Checks-Effects-Interactions pattern is violated. During the functions, some state variables are updated, and events are emitted after the external calls.

This lowers code quality and can lead to reentrancy attacks.

Path: ./contracts/preSale.sol : withdrawToken();

Recommendation: Refactor code to fit CEI pattern.

Status: Fixed (revised commits: a893ff6, 31fc993)

■■ Medium

1. Using SafeMath in Solidity ^0.8.0

Functionality of SafeMath library is prebuilt in Solidity ^0.8.0, this makes its import redundant. This library can be removed to save Gas.

This will lower Gas taxes.

Path: ./contracts/*

Recommendation: Remove redundant library.

Status: Fixed (revised commits: a893ff6, 31fc993)

2. Unoptimized Loops Usage

Contracts use loops without optimization.

This leads to higher Gas taxes.

Paths: ./contracts/preSale.sol : setRoundDetails(); ./contracts/preSale.sol : withdrawToken(); ./contracts/preSale.sol : setExternalAllocation();

www.hacken.io

Recommendation: Cache arrays in a loop, save state variables to local memory, iterate the loop and save changes to the state after the loop finishes, avoid using external calls inside a loop.

Status: Fixed (revised commits: a893ff6, 31fc993)

3. Potential Invalid Values

Function *setVestingPeriod()* is setting vesting begin time but does not check the correlation with the current time. Due to this, past time can be mistakenly set as a vesting begin time.

Path: ./contracts/preSale.sol : setVestingPeriod()

Recommendation: Implement time checks.

Status: Fixed (revised commits: a893ff6, 31fc993)

◼ Low

1. State Variables Can Be Declared Constant

Variables whose value is hardcoded in the contract and never changed should be declared constant.

This will lower Gas taxes.

Path: ./contracts/ReduxToken.sol : supply

Recommendation: Remove redundant library.

Status: Fixed (revised commits: a893ff6, 31fc993)

2. 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.

Path: ./contracts/OwnerWithdrawable.sol

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

Status: Fixed (revised commits: a893ff6, 31fc993)

3. Public Functions that Can Be Declared External

Public functions that are never called in the contract should be declared external to save Gas.

This will lower Gas taxes.

Paths: ./contracts/OwnerWithdrawable.sol: withdrawCurrency(); ./contracts/OwnerWithdrawable.sol: getCurrencyBalance(); ./contracts/preSale.sol: getTokensBought(); ./contracts/preSale.sol: getRoundsParticipated(); ./contracts/preSale.sol: getTokensPerRound(); ./contracts/preSale.sol: getClaimedTokensPerRound(); ./contracts/preSale.sol:

www.hacken.io

getMonthlyVestingClaimed();   ./contracts/preSale.sol:
getTotalClaimedTokens();

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

Status: Fixed (revised commits: a893ff6, 31fc993)

4. State Variables Default Visibility

Variable`s *saleToken* visibility is not specified. Specifying state variables` visibility helps to catch incorrect assumptions about who can access the variable.

This makes the contract`s code quality and readability higher.

Path: ./contracts/preSale.sol: saleToken

Recommendation: Specify variables as public, internal, or private. Explicitly define visibility for all state variables.

Status: Fixed (revised commits: a893ff6, 31fc993)

5. 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.

Path: ./contracts/preSale.sol: constructor()

Recommendation: Implement zero address checks.

Status: Fixed (revised commits: a893ff6, 31fc993)

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed by the best industry practices at the date 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 to 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, Consultant cannot guarantee the explicit security of the audited smart contracts.