SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: VYNKSAFE
Date: May 31st, 2022

This document 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 containing confidential information can be used internally by the Customer, or it can be disclosed publicly after all vulnerabilities are fixed — upon a decision of the Customer.

Document

Name	Smart Contract Code Review and Security Analysis Report for VYNKSAFE.
Approved By	Evgeniy Bezuglyi | SC Department Head at Hacken OU
Type	BEP-20 token; Staking
Platform	EVM
Language	Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Website	https://vynksafe.com
Timeline	06.05.2022 - 25.05.2022
Changelog	09.05.2022 - Initial Review 20.05.2022 - Second Review 25.05.2022 - Third Review 31.05.2022 - Fourth Review

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by VYNKSAFE (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:

-

Commit:

-

Technical Documentation:

Type: Whitepaper (partial functional requirements provided)
Link: https://vynkchain.org/assets/vc/vynkgroup-wp.pdf

Type: Technical description
Link:
https://docs.google.com/document/d/14MLcG-q_bfg4xdQk6CCTQv2hOFwpvEmw-oM44avjiR0/edit

Type: Functional requirements
Link:
https://docs.google.com/document/d/14MLcG-q_bfg4xdQk6CCTQv2hOFwpvEmw-oM44avjiR0/edit

JS tests: No

Deployed Contracts Addresses:

https://testnet.bscscan.com/address/0xC6Dd7aF3C5B1b36Ba5bD1bE1a07d2297d2C7448a#code

Contracts:

File: ./contracts/main.sol
SHA3: 669855257b3e2b0650dd250fce5df47376e673523405a86dcda35302

Second review scope

Repository:

-

Commit:

-

Technical Documentation:

Type: Whitepaper (partial functional requirements provided)
Link: https://vynkchain.org/assets/vc/vynkgroup-wp.pdf

Type: Technical description
Link:
https://docs.google.com/document/d/14MLcG-q_bfg4xdQk6CCTQv2hOFwpvEmw-oM44avjiR0/edit

Type: Functional requirements
Link:
https://docs.google.com/document/d/14MLcG-q_bfg4xdQk6CCTQv2hOFwpvEmw-oM44avjiR0/edit

Deployed Contracts Addresses:
https://testnet.bscscan.com/address/0x56DB41a922b59Fd1b5C5dEd0250a1856aC653d87#code

Contracts:
File: ./contracts/main.sol
SHA3: 8ab65e51a1dbdf15bbc6a8dc02bfd29e0161374caf631dde20f33f59

Third review scope

Repository:
-

Commit:
-

Technical Documentation:
Type: Whitepaper (partial functional requirements provided)
Link: https://vynkchain.org/assets/vc/vynkgroup-wp.pdf

Type: Technical description
Link:
https://docs.google.com/document/d/14MLcG-q_bfg4xdQk6CCTQv2hOFwpvEmw-oM44avjiR0/edit

Type: Functional requirements
Link:
https://docs.google.com/document/d/14MLcG-q_bfg4xdQk6CCTQv2hOFwpvEmw-oM44avjiR0/edit

Deployed Contracts Addresses:
https://testnet.bscscan.com/address/0x3C67eB4e3056E96E6AB2e2a3112c0C381D66768B#code

Contracts:
File: ./contracts/main.sol
SHA3: ac8491acbf0e76ed09573f44138d7c1f076062ab4ebad5de11175dd3

Fourth review scope

Repository:
-

Commit:
-

Technical Documentation:
Type: Whitepaper (partial functional requirements provided)
Link: https://vynkchain.org/assets/vc/vynkgroup-wp.pdf

Type: Technical description
Link:
https://docs.google.com/document/d/14MLcG-q_bfg4xdQk6CCTQv2hOFwpvEmw-oM44avjiR0/edit

Type: Functional requirements
Link:
https://docs.google.com/document/d/14MLcG-q_bfg4xdQk6CCTQv2hOFwpvEmw-oM44avjiR0/edit

Deployed Contracts Addresses:
https://bscscan.com/address/0x47e4035A62c3E469eC37f71D7A73AA4e37111B7f

Contracts:
File: ./contracts/BUSDVYNCSTAKE.sol
SHA3: 5fbb6ae2bde1542438e3453aec5d9aabae779170603ab09b3994706e

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to assets loss or data manipulations.
High	High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g., public access to crucial functions
Medium	Medium-level vulnerabilities are important to fix; however, they cannot lead to assets loss or data manipulations.
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that cannot have a significant impact on execution

Executive Summary

The score measurement details can be found in the corresponding section of the methodology.

Documentation quality

The Customer provided full documentation with functional requirements. The total Documentation Quality score is **10** out of **10**.

Code quality

The total CodeQuality score is **4** out of **10**. Unit tests were not provided. The code has duplications.

Architecture quality

The architecture quality score is **5** out of **10**. The project has mostly clean and clear architecture. The project has no configured development environment.

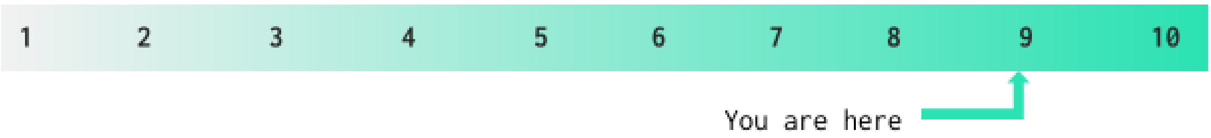
Security score

As a result of the audit, security engineers found **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: **8.9**



Checked Items

We have audited provided smart contracts for commonly known and more specific vulnerabilities. Here are some of the items that are 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 destroyed until 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
Uninitialized Storage Pointer	SWC-109	Storage type should be set explicitly if the compiler version is < 0.5.0.	Not Relevant
Assert Violation	SWC-110	Properly functioning code should never reach a failing assert statement.	Not Relevant
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 it is required.	Not Relevant

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	Signed messages should always have a unique id. A transaction hash should not be used as a unique id.	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.	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.	Not Relevant
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.	Passed
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
Repository Consistency	**Custom**	The repository should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Failed
Tests Coverage	**Custom**	The code should be covered with unit tests. Tests coverage should be 100%, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Failed

System Overview

VYNKSAFE is a staking contract, which allows to stake BUSD tokens, which are automatically swapped to the liquidity tokens.
The contract gets price data for the swap amount calculation using an off-chain source.

Privileged roles

The owner is allowed to:

- change `treasury` contract address. This contract stores rewards.
- change `data` contract address.
- set compound start time.
- transfer tokens from staking contract to any address, except liquidity tokens.

Risks

- The Customer did not provide unit tests and a development environment for the code. The risk of unexpected issues is higher because proper validation of edge cases is impossible.
- GetDataInterface implementation where token prices come from could not be verified in the audit scope.

Findings

🟥🟥🟥🟥 Critical

No critical issues were found.

🟨🟨🟨 High

1. **Incorrect unstaking function logic.**

   If the user either calls `stake` or `unStake` function, `totalClaimedReward` amount is set to `0`, so the user can claim no more tokens.

   This can lead to the leak of funds from the contract.

   Contracts: main.sol

   Function: unStake, stake

   Recommendation: Do not reset `totalClaimedReward` amount if the user does not withdraw all the funds.

   Status: Fixed
   (testnet add:0x56DB41a922b59Fd1b5C5dEd0250a1856aC653d87)

2. **Flash loan attack.**

   VYNC-BUSD exchange rate is calculated depending on the current rate received from a DEX. The exchange rate can be easily manipulated using flash loans. The rewards amount is calculated according to the exchange rate.

   More rewards can be received, and the VYNC exchange rate will be dropped.

   Contracts: main.sol

   Functions: vyncPerBusd

   Recommendation: Use oracles to receive an exchange rate.

   Status: Fixed
   (testnet address: 0x3C67eB4e3056E96E6AB2e2a3112c0C381D66768B)

3. **Sandwich attack.**

   It is possible to monitor users' transactions into the pool of transactions and front-run this transaction.

   This will allow to manipulate the VYNC/BUSD ratio and get an advantage out of plain users' transactions.

   Contracts: main.sol

   Functions: stake, unStake

   Recommendation: Minimum and maximum bounds for the swap functionality should be set from an off-chain source.

Status: Fixed
(testnet address: 0x3C67eB4e3056E96E6AB2e2a3112c0C381D66768B)

🟩🟩 Medium

1. **Swap procedures take fees.**

   The contract benefits from PanckeSwap's swap functionality. It is, by design, takes fees from the users.

   This causes the receiver to get less than the desired amount every time.

   Contracts: main.sol

   Function: cPendigReward, swapBusdtoVync

   Recommendation: Notify users about these fees.

   Status: Fixed
   (testnet address: 0x56DB41a922b59Fd1b5C5dEd0250a1856aC653d87)

2. **Unchecked transfer return value.**

   The contract widely interacts with ERC20 contract by calling the `transfer` and `transferFrom` functions, but the return value is never checked.

   Contracts: main.sol

   Function: stake, unStake, claim

   Recommendation: Implement return value checks.

   Status: Fixed
   (testnet address: 0x56DB41a922b59Fd1b5C5dEd0250a1856aC653d87)

🟦 Low

1. **Typical library function declaration.**

   The contract has the typical `sqrt` math function that may be loaded from the Math library.

   This may lead to unnecessary Gas usage during the contract deployment.

   Contracts: main.sol

   Function: sqrt

   Recommendation: use the library function.

   Status: Fixed
   (testnet address: 0x56DB41a922b59Fd1b5C5dEd0250a1856aC653d87)

2. **Redundant function call.**

   The contract inside the `unStake` function may call twice the `compoundedReward` function.

The `returnData` function can return both the compound rate and the APR value at once. There is no need to call this function twice to get them separately.

In the `compoundedReward` function is a local variable that holds the cPrendigReward value. There is no need for a separate function call while computing _compundedReward value.

These may lead to unnecessary Gas usage during the calls to the `unStake` function.

Contracts: main.sol

Function: unStake, compoundedReward, pendingReward, lastCompoundedReward

Recommendation: Delete these function calls and update the functions accordignly.

Status: Fixed
(testnet address: 0x56DB41a922b59Fd1b5C5dEd0250a1856aC653d87)

3. **Redundant use of SafeMath.**

Since Solidity v0.8.0, the overflow/underflow check is implemented on the language level - it adds the validation to the bytecode during compilation.

There is no need to use the SafeMath library.

Contracts: main.sol

Recommendation: Remove the SafeMath library.

Status: Fixed
(testnet address: 0x56DB41a922b59Fd1b5C5dEd0250a1856aC653d87)

4. **Missing event emitting.**

Events for critical state changes (e.g. owner and other critical parameters) should be emitted for tracking things off-chain.

Contracts: main.sol

Function: set_treasuryAddress, set_data

Recommendation: Create and emit related events.

Status: Fixed
(testnet address: 0x56DB41a922b59Fd1b5C5dEd0250a1856aC653d87)

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.

Contracts: main.sol

Function: set_treasuryAddress, set_data

Recommendation: Implement zero address validations.

Status: Fixed
(testnet address: 0x56DB41a922b59Fd1b5C5dEd0250a1856aC653d87)

6. **Redundant code block.**

The `lpAmountNeeded` value is being set conditionally, and the value is being checked with a require statement. However, in both conditions, the value being set is the same.

It is redundant to use that code block.

Contracts: main.sol

Function: unstake

Recommendation: Delete redundant code block.

Status: Fixed
(testnet address: 0x56DB41a922b59Fd1b5C5dEd0250a1856aC653d87)

7. **Floating pragma.**

The project uses floating pragmas ^0.8.0.

Contracts: main.sol

Function: -

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

Status: Fixed
(testnet address: 0x56DB41a922b59Fd1b5C5dEd0250a1856aC653d87)

8. **Use of hardcoded values.**

Hard-coded values are used in computations and comparisons.

Contracts: main.sol

Function: unstake, cPendingReward, compundedReward, vyncPerUSD, vyncRateInBusd, calculateSwapInAmount, getLPTokenAmount1

Recommendation: Move these values to constants.

Status: Fixed
(testnet address: 0x56DB41a922b59Fd1b5C5dEd0250a1856aC653d87)

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 audit makes no statements or warranties on the security of the code. It also cannot be considered a 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.

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 audit cannot guarantee the explicit security of the audited smart contracts.