HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Cirus
Date: December 7th, 2022

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 Cirus
Approved By	Evgeniy Bezuglyi | SC Audits Department Head at Hacken OU
Type	Staking
Platform	EVM
Language	Solidity
Methodology	Link
Changelog	30.11.2022 - Initial Review 07.12.2022 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 Cirus (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/alwaysaugust/cirus-dashboard
Commit	47082a4bab524617c6e59bcd0d7f07d1364a1457
Functional Requirements	https://support.cirusfoundation.com/en/knowledge/cirus-token-hub#cirus-community-staking-pool
Technical Requirements	Provided in NatSpec comments
Contract	File: ./src/contracts/StakingV2Contract.sol SHA3: f0d774c1687aae8435ef809d1ed9b91f1f43a293a61df5a5a2474ed9ca34b0a6

Second review scope

Repository	https://github.com/alwaysaugust/token-hub-contract
Commit	860e1a2e15a10292dfd14b4cbbee5d2b7d667741
Contract	File: ./CirusStakingV3Contract.sol SHA3: 0db6aafdf0c94b290962d36a67e48d66214d226406c5a3d038488c5033cfc002

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 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Low	Low vulnerabilities are related to outdated and unused code or minor gas optimization.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

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

Code quality

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

Test coverage

Tests do not exist.

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**.

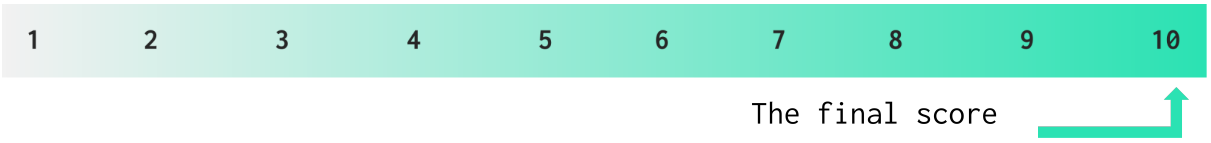

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
28 November 2022	8	2	2	2
7 December 2022	1	0	0	0

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 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.	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.	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
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.	Not Relevant
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 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

System Overview

CirusStakingV3Contract – a contract that rewards users for staking their tokens. APY depends on rewardPerMonth variable and the total stake amount. Inherited from Ownable, Pausable, ReentrancyGuard.

It has the following attributes:

- _cirusToken: Represent Cirus Token.
- _startTime: The reward starts distribution. It can set the current time when deploying the contract.
- _fee: Percent of fee goes to _feeCollectAddr from 0 to 100.
- _rewardPerMonth: Total monthly reward amount, for example: if needed to distribute 50,000 Cirus Tokens per month among stakers, set this value to 50,000 * 10 ** 18.
- _rewarderAddr: Rewards are distributed from this address.
- _feeCollectAddr: Fees are transferred to this address when the user deposits tokens.

Privileged roles

- The owner of the contract can pause and unpause the contract.

Findings

■■■■ Critical

1. Insufficient Balance

In the *emergencyWithdraw()* function sets the *user.amount* to 0 and then uses the same value in transfer.

A user will receive 0 tokens, and funds will be locked on the contract.

Path: ./contracts/StakingV2Contract.sol : emergencyWithdraw()

Recommendation: transfer funds via a local variable

Status: Fixed (Revised commit: 860e1a2)

2. Data Consistency

The *totalStakedAmount* variable stores the total staked amount. When a user withdraws tokens via the *withdraw* function, the *totalStakedAmount* value is decreased, but the *emergencyWithdraw* lacks the value decrease.

In case of the *emergencyWithdraw* function call, the *totalStakedAmount* value will remain higher than the actual balance of the contact.

Path: ./contracts/StakingV2Contract.sol : emergencyWithdraw()

Recommendation: subtract the withdrawn value from the *totalStakedAmount*.

Status: Fixed (Revised commit: 860e1a2)

■■■ High

1. Undocumented Behaviour

The project deducts fees. The documentation does not contain information regarding fees.

The behavior is not described in the available documentation.

Path: ./contracts/StakingV2Contract.sol

Recommendation: document this functionality in the public documentation.

Status: Fixed (Revised commit: 860e1a2)

2. Invalid Hardcoded Value

The project has hardcoded important values such as wallet's private key, infuraId and etherscan API key in the environment setup files.

Path: ./hardhat.config.js

Recommendation: to store and use secrets, use .env files excluded from git commits.

Status: Fixed (Revised commit: 860e1a2)

Medium

1. Best Practice Violation

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

Path: ./contracts/StakingV2Contract.sol : deposit()

Recommendation: implement the *deposit()* function according to the Checks-Effects-Interactions pattern.

Status: Fixed (Revised commit: 860e1a2)

2. Insufficient Gas Model

Since Solidity v0.8.0, the overflow/underflow check is implemented via ABIEncoderV2 on the language level - it adds the validation to the bytecode during compilation.

There is no need to use the SafeMath library.

Path: ./contracts/StakingV2Contract.sol

Recommendation: remove redundant import.

Status: Fixed (Revised commit: 860e1a2)

Low

1. Floating Pragma

The project uses floating pragma 0.8.11.

Path: ./contracts/StakingV2Contract.sol

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

Status: Fixed (Revised commit: 860e1a2)

2. State Variables Can Be Declared Immutable

mulDecimal variable declared with value and never changed. This variable can be declared immutable. This will lower the Gas taxes.

Path: ./contracts/StakingV2Contract.sol

Recommendation: declare mentioned variables as immutable.

Status: Fixed (Revised commit: 860e1a2)

3. State Variables Can Be Declared Constant

_cirusToken can be declared as constant. This will lower the Gas taxes.

Path: ./contracts/StakingV2Contract.sol

Recommendation: declare mentioned variables as constant.

Status: Fixed (Revised commit: 860e1a2)

4. **Unused Imports**

Pausable contract important from OpenZeppelin but never use it.

The code has an unused console log import; it should be removed.

The code contains commented out imports that should be removed in the final code.

Path: ./contracts/StakingV2Contract.sol

Recommendation: remove redundant imports.

Status: Fixed (Revised commit: 860e1a2)

5. **Missing Zero Address Validation**

Address parameters are being used without checking against the possibility of 0x0.

Paths: ./contracts/StakingV2Contract.sol : pendingReward()

./contracts/StakingV2Contract.sol : constructor()

Recommendation: implement zero address checks.

Status: Fixed (Revised commit: 860e1a2)

6. **State Variables Default Visibility**

The explicit visibility makes it easier to catch incorrect assumptions about who can access the variable.

Variables *lastRewardTime* and *accCirusPerShare* 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/StakingV2Contract.sol

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

Status: Fixed (Revised commit: 860e1a2)

7. **Event Names Mismatch**

Event names contradict events' value or value usage purposes.

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

The events' *Withdrawed*, *EmergencyWithdrawed* name and value contradict.

This makes the contract harder to read.

Path: ./contracts/StakingV2Contract.sol

Recommendation: change event names to fit their value.

Status: Reported

8. **Use of Hard-Coded Values**

Using hardcoded values in the computations and comparisons is not the best practice.

Hard-coded values are used in computations.

Paths: ./contracts/StakingV2Contract.sol: estimatedAPY()

./contracts/StakingV2Contract.sol: updateRewardAmount()

./contracts/StakingV2Contract.sol: deposit()

./contracts/StakingV2Contract.sol: constructor()

Recommendation: convert these variables into constants.

Status: Fixed (Revised commit: 860e1a2)

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.