HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: MyCowrie
Date: July 21th, 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 MyCowrie
Approved By	Evgeniy Bezuglyi | SC Audits Department Head at Hacken OU
Type	Staking, Vesting
Platform	EVM
Network	Ethereum, BSC
Language	Solidity
Methods	Manual Review, Automated Review, Architecture review
Website	https://mycowrie.org/
Timeline	17.06.2022 - 21.07.2022
Changelog	28.06.2022 - Initial Review 14.07.2022 - Second Review 15.07.2022 - Third Review 21.07.2022 - Fourth 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 MyCowrie (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/safcoin-project/CowrieProject
Commit:
2e2166126e04f8c97840f46e30447ae8280f711e
Technical Documentation:
Type: Whitepaper
Attached to the Audit scope

Type: Functional requirements
Attached to the Audit scope

Integration and Unit Tests: No
Deployed Contracts Addresses: No
Contracts:
File: ./contracts/Migrations.sol
SHA3: f38ad4185f0fa410f3427a0bae9195f29bf1c8806f1a019cc727d7c39b53811d

File: ./contracts/Staking.sol
SHA3: b3ebc0c958f63fc5d3de3ba63652861fc7b9c1eb980f8e65221ecc41cd8a9e28

File: ./contracts/Token.sol
SHA3: b1e3d53cbc470c842c09e77adac328c22d9298fe94dbe0de4187f19b9d168ddc

File: ./contracts/Vesting.sol
SHA3: 141739bae8c219ed19375501eaedadff03beba2112f9bc532bc40cd33e22d153

Second review scope

Repository:
https://github.com/safcoin-project/CowrieProject
Commit:
9501b9b561bf86bffd88d8dc304cfbff7730361e
Technical Documentation:
Type: Whitepaper
Attached to the Audit scope

Type: Functional requirements
Attached to the Audit scope

Integration and Unit Tests: No
Deployed Contracts Addresses: No
Contracts:
File: ./contracts/Migrations.sol
SHA3: f38ad4185f0fa410f3427a0bae9195f29bf1c8806f1a019cc727d7c39b53811d

File: ./contracts/Staking.sol

SHA3: 6970bb7997ac548c34119f7642ee4b03f1cd690530d0592bbaa5fe1995210db9

File: ./contracts/Token.sol
SHA3: 393dc76c175e049f677d64dca9138ecc7c17af556ee9c945df75216bad9428a2

Third review scope

Repository:
https://github.com/safcoin-project/CowrieProject
Commit:
2e544b78416a5241253d587da89e767b55c1874e
Technical Documentation:
Type: Whitepaper
Attached to the Audit scope

Type: Functional requirements
Attached to the Audit scope

Integration and Unit Tests: No
Deployed Contracts Addresses: No
Contracts:
File: ./contracts/Migrations.sol
SHA3: f38ad4185f0fa410f3427a0bae9195f29bf1c8806f1a019cc727d7c39b53811d

File: ./contracts/Staking.sol
SHA3: 939650e98735438415d9b797733bbbf69e8138005bcb08cd9e7c3f918a76a9a2

File: ./contracts/Token.sol
SHA3: 393dc76c175e049f677d64dca9138ecc7c17af556ee9c945df75216bad9428a2

File: ./contracts/Vesting.sol
SHA3: 4d70342368d3560f7a245b4a7960bbd0872d36dd66d57bbe04c32a64ac34577e

Fourth review scope

Repository:
https://github.com/safcoin-project/CowrieProject
Commit:
502542a2563eeb42e1d986ed82174486f2227d6b
Technical Documentation:
Type: Whitepaper
Attached to the Audit scope

Type: Functional requirements
Attached to the Audit scope

Integration and Unit Tests: No
Deployed Contracts Addresses: No
Contracts:
File: ./contracts/Migrations.sol
SHA3: f38ad4185f0fa410f3427a0bae9195f29bf1c8806f1a019cc727d7c39b53811d

File: ./contracts/Staking.sol
SHA3: 939650e98735438415d9b797733bbbf69e8138005bcb08cd9e7c3f918a76a9a2

File: ./contracts/Token.sol
SHA3: 393dc76c175e049f677d64dca9138ecc7c17af556ee9c945df75216bad9428a2

File: ./contracts/Vesting.sol
SHA3: 43ddcbdb29af1caf168531b1bca85932dc94bde529fee638b0187ac7b365938b

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 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.

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 methodology.

Documentation quality

The total Documentation Quality score is **6** out of **10**. Functional requirements and whitepaper provided. A technical description is not provided.

Code quality

The total CodeQuality score is **6** out of **10.** Tests were not provided.

Architecture quality

The architecture quality score is **7** out of **10**. Development environment configuration instructions are missing.

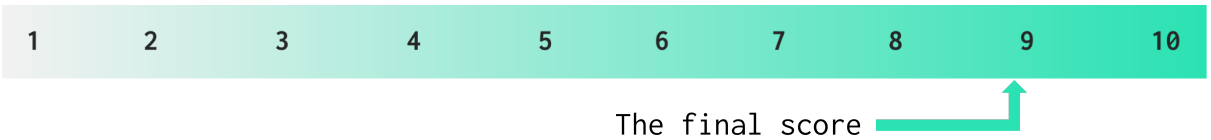
Security score

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: **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.	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.	Passed
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.	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.	Passed
Race Conditions	SWC-114	Race Conditions and Transactions Order Dependency should not be possible.	Not Relevant
Authorization	SWC-115	tx.origin should not be used for	Passed

through tx.origin		authorization.	
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 identifier 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.	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	Failed

		data stored on the contract. There should not be any cases when execution fails due to the block Gas limit.	
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.	Failed
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.	Failed
Stable Imports	**Custom**	The code should not reference draft contracts, that may be changed in the future.	Passed

System Overview

CowrieProject is a system, which represents ERC20 token together with Vesting, Staking contracts for this token.

- *Token* – simple ERC-20 token that mints all initial supply to a deployer. Additional minting is allowed.
  It has the following attributes:
  - Initial Supply
- *Staking* – a contract that rewards users for staking tokens in different pools. APY depends on the pool percent, tokens provided by the staker, and staking duration.
- Vesting – a contract that provides minting to the beneficiaries upon each release. Each beneficiary has its percentage of total vesting tokens, which will be minted to the beneficiary according to how many years have passed. Totally 27 years. Additional SAPD tokens can be minted.

Privileged roles

- The admin can:
  - set minimum and maximum amount can be deposited to the Staking.
  - pause/unpause contract.
  - fund pool with rewards.
  - withdraw any ERC20 accidentally sent to contract apart from reward token.
  - create a pool.
- The owner can:
  - add vesting address.
  - add a beneficiary.
  - add beneficiary subwallets.
  - add admins.
  - update SAPD officer address.
  - update beneficiary valid status.
  - release tokens.
- The SAPD officer can
  - release SAPD tokens.
- The vesting can
  - mint new tokens.

Risks

- The total capped supply of the token could not be verified before the deployment.
- If the number of sub wallets reaches a very large number, the owner may not be able to release the sub wallet tokens or update the beneficiary address due to Gas limitations.

Findings

Critical

No critical severity issues were found.

High

1. **Highly permissive owner access**

   In the *Staking* contract, the admin has the ability to change the APY and duration anytime. Moreover, the admin can remove the pool.

   This may affect user rewards up to the case the user never gets rewards.

   Contract: Staking.sol

   Functions: editPool, removePool

   Recommendation: Do not allow those parameters to be rescheduled.

   Status: Fixed (Revised commit: 9501b9b561bf86bffd88d8dc304cfbff7730361e)

2. **Insufficient rewards balance**

   Rewards balances are not separated from staking balances.

   Rewards of one user could be paid on behalf of funds staked by other users.

   Contract: Staking.sol

   Recommendation: Separate staking and rewards balances and ensure that all users will be able to withdraw their deposits whenever they want to.

   Status: Fixed (Revised commit: 9501b9b561bf86bffd88d8dc304cfbff7730361e)

3. **Highly permissive owner access**

   The beneficiary's percentage can be changed. This may affect the number of tokens the user can claim upon vesting event. Moreover, totalReleasableToken is not getting recalculated upon changing percentage.

   Contract: Vesting.sol

   Function: updateBeneficiaryPercentages

   Recommendation: Do not allow those parameters to be rescheduled. Otherwise, clearly mention it in whitepaper documentation and consider totalReleasableToken recalculation.

   Status: Fixed (Revised commit : 2e544b78416a5241253d587da89e767b55c1874e)

Medium

1. **Potential DoS**

   The function iterates over all pools.

   DoS is possible if the number of pools is large enough.

   Contract: Staking.sol

   Function: getTotStakedAndAlloc

   Recommendation: Calculate *totStakedAmount* and *totAlloc* upon changing *pool.lockedLimit, pool.stakedAmount* or limit the maximum number of pools can be iterated. Otherwise, make sure this is not an issue.

   Status: Mitigated (with Customer notice) (Revised commit: 9501b9b561bf86bffd88d8dc304cfbff7730361e)

2. **Mixed event parameters**

   Event *Withdraw* parameters *reward* and *totalWithdraw* are mixed up.

   Therefore *Withdraw* events will serve as confusing information.

   Contract: Staking.sol

   Function: withdraw

   Recommendation: Swap the parameters.

   Status: Fixed (Revised commit: 9501b9b561bf86bffd88d8dc304cfbff7730361e)

3. **Wrong event**

   Event *NewPool* is emitted upon editing pool. The same event is emitted upon pool creation.

   Therefore it is harder to distinguish between pool creation and editing.

   Contract: Staking.sol

   Function: editPool

   Recommendation: Add a specific event for pool editing.

   Status: Fixed (Revised commit: 9501b9b561bf86bffd88d8dc304cfbff7730361e)

4. **Checks-Effects-Interactions pattern violation**

   In functions *'deposit'*, *'depositFor'*,*'restake'*; require statements are used after the state changes.

   Contract: Staking.sol

   Functions: deposit, depositFor, restake

Recommendation: Make sure all internal state changes are performed after the require statements.

Status: Fixed (Revised commit: 2e544b78416a5241253d587da89e767b55c1874e)

5. **Potential DoS**

The function iterates over all beneficiaries. Possible DoS if the number of beneficiaries is large enough.

Contract: Vesting.sol

Function: getAllBeneficiaryDetails

Recommendation: Review and fix the logic so that it will not be needed to iterate over all beneficiaries. For example, calculate releasable tokens upon changing parameters from which their value is related.

Status: Mitigated (with Customer notice) (Revised commit: 502542a2563eeb42e1d986ed82174486f2227d6b)

6. **Potential DoS**

The function iterates over all beneficiary subwallets. Possible DoS if the number of subwallets is large enough. Line number 200.

Contract: Vesting.sol

Function: updateBeneficiaryAddress

Recommendation: Review and fix the logic so that it will not be needed to iterate over all subwaletts. Otherwise, limit the number of subwallets beneficiary can have.

Status: Mitigated (with Customer notice) (Revised commit: 502542a2563eeb42e1d986ed82174486f2227d6b)

7. **Redundant reentrancy guard**

Contract ReentrancyGuard extension is redundant as long as Vesting contract never sends any Ether.

Contract: Vesting.sol

Functions: releaseLinearTokens, releaseSAPDTokens, releaseSubwalletTokens

Recommendation: Remove redundant extension.

Status: Fixed (Revised commit: 2e544b78416a5241253d587da89e767b55c1874e)

8. **Structure property never checked**

Property toReleaseSubWallets of the beneficiaryDetails structure established in a few places but never used.

Therefore it makes no sense to have this property. It should be used in the releaseSubwalletTokens function to release tokens on the beneficiary sub wallets and do not perform code of this function if this property is not set.

Contract: Vesting.sol

Functions: releaseLinearTokens, releaseSubwalletTokens, releaseSAPDTokens.

Recommendation: Review and fix the logic.

Status: Fixed (Revised commit: 2e544b78416a5241253d587da89e767b55c1874e)

Low

1. Floating pragma

The contracts use floating pragma ^0.8.0, ^0.8.7.

Contracts: all

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

Status: Fixed (Revised commit: 9501b9b561bf86bffd88d8dc304cfbff7730361e)

2. Variables can be declared as immutable

No setter functions for *companyWallet* and *token* state variables. It can be declared immutable.

Immutable variables decrease storage costs.

Contract: Staking.sol

Recommendation: Change *companyWallet* and *token* variables to immutable.

Status: Fixed (Revised commit: 9501b9b561bf86bffd88d8dc304cfbff7730361e)

3. Using SafeMath

SafeMath is generally not needed starting with Solidity 0.8+

Contract: Token.sol

Recommendation: Remove SafeMath.

Status: Fixed (Revised commit: 9501b9b561bf86bffd88d8dc304cfbff7730361e)

4. Confusing modifier name

Modifier name is *isFinished* while it checks the opposite state that *staker.isFinished == false*.

Contract: Staking.sol

Function: modifier isFinished

Recommendation: Rename to a more appropriate name, for example, *isNotFinished, isAvailable*.

Status: Fixed (Revised commit: 9501b9b561bf86bffd88d8dc304cfbff7730361e)

5. **Confusing function name**

The function name is *addVestingAddress* while it sets the address and does not add a new one.

Contract: Staking.sol

Function: addVestingAddress

Recommendation: Rename the function to the appropriate name, for example, *setVestingAddress*.

Status: Fixed (Revised commit: 9501b9b561bf86bffd88d8dc304cfbff7730361e)

6. **Style guide violation**

Contracts do not follow the Solidity code style guide.

Contracts: all

Recommendation: Follow the official Solidity code style guide.

Status: Fixed (Revised commit: 502542a2563eeb42e1d986ed82174486f2227d6b)

7. **Redundant use of SafeMath**

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.

File: ./contracts/Vesting.sol

Contract: Vesting.sol

Function: -

Recommendation: Remove SafeMath.

Status: Fixed (Revised commit: 502542a2563eeb42e1d986ed82174486f2227d6b)

8. **Functions that can be declared as external**

To save Gas, public functions that are never called in the contract should be declared as external.

Contract: Vesting.sol

Functions:addBeneficiary,updateBeneficiaryPercentages, releaseLinearTokens,addBeneficiarySubWallets, updateSubWalletValidStatus, releaseSubwalletTokens,releaseSAPDTokens, getBeneficiaryDetails, getAllBeneficiaryDetails, getBeneficiaries, getBeneficiarySubWallets

Recommendation: Use the external attribute for functions never called from the contract.

Status: Fixed (Revised commit: 2e544b78416a5241253d587da89e767b55c1874e)

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.