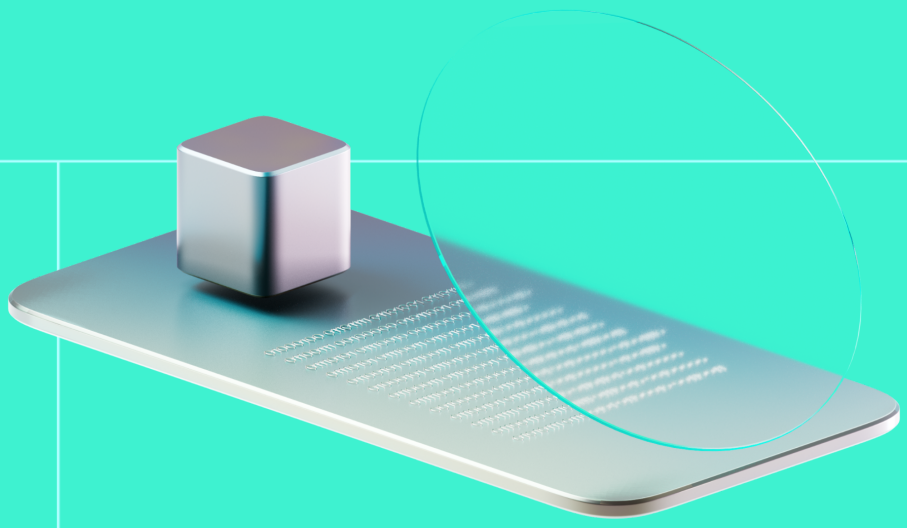
HACKEN

Smart Contract Code Review And Security Analysis Report



Customer: Coinbuck

Date: 22 Dec, 2023

Hacken OU
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia

We thank Coinbuck for allowing us to conduct a Smart Contract Security Assessment. This document outlines our methodology, limitations, and results of the security assessment.

Coinbuck is an ERC-20 token.

Platform: EVM

Language: Solidity

Tags: ERC-20

Timeline: 04.12.2023 - 22.12.2023

Methodology: Link

Last review scope

Repository	https://github.com/Coinbuck/Contracts/tree/development
Commit	f8ba87

View full scope

Hacken OU
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia

Audit Summary

10/10	9/10	91.67%	10/10
Security score	Code quality score	Test coverage	Documentation quality score

Total: 9.5/10

The system users should acknowledge all the risks summed up in the risks section of the report.

1	1	0	0
Total Findings	Resolved	Acknowledged	Mitigated

Findings by severity	Findings Number	Resolved	Mitigated	Acknowledged
Critical	0	0	0	0
High	0	0	0	0
Medium	1	1	0	0
Low	0	0	0	0

Hacken OU
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia

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 Coinbuck
Audited By	Carlo Parisi | SC Lead Auditor at Hacken OÜ Roman Tiutiun| SC Auditor at Hacken OÜ
Approved By	Przemyslaw Swiatowiec | SC Audits Expert at Hacken OÜ
Changelog	08.12.2023 – Preliminary Report 20.12.2023 – Final Report

Hacken OU
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia

Introduction

Hacken OÜ (Consultant) was contracted by Coinbuck (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.

System Overview

Coinbuck is a staking protocol with the following contracts:

- BuckToken — ERC-20 token that mints all initial supply to a deployer. Additional minting is not allowed.

  It has the following attributes:

  - Decimals: 18
  - Total supply: 1B tokens.

Privileged roles

- The owner of the *buck.sol* contract can arbitrarily add, remove from a backlist, enable or disable the backlist.

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**.

- Functional requirements are provided.
- NatSpec is sufficient.
- Technical description is provided.

Code quality

The total Code Quality score is **9** out of **10**.

- The code is well structured.

- Best practice violations (I01, I02, I04)

Test coverage

Code coverage of the project is **91.67%** (branch coverage).

Security score

As a result of the audit, the code does not contain any severity 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**. The system users should acknowledge all the risks summed up in the risks section of the report.

Risks

- The system exhibits a high degree of centralization, creating a risk of funds becoming trapped in blacklisted addresses. Additionally, the entire token supply is minted to the contract deployer, consolidating control and potentially raising concerns about equitable distribution.

Findings

Critical

No critical severity issues were found

High

No high severity issues were found.

Medium

M01. Unrestricted regain due to deviation from the standard ownership renouncement process.

Impact	Medium
Likelihood	Medium

The *Ownable* contract contains a security vulnerability within the *updateOwner()* function. Presently, when *updateOwner()* is invoked, it verifies whether the message sender, *_msgSender(),* corresponds to the *_pendingOwner* and is a non-zero address.

Consequently, utilizing the *updateOwner()* function leads to the continual assignment of the *_pendingOwner* variable, persisting even after the execution of the *renounceOwnership()* function.

```
function renounceOwnership() external virtual onlyOwner {
        emit OwnershipTransferred(_owner, address(0));
        _owner = address(0);
    }

function updateOwner() external {
        require(_msgSender() == _pendingOwner, "Not pendind owner");
        require(
            _msgSender() != address(0),
            "Ownable: new owner is the zero address"
        );
        emit OwnershipTransferred(_owner, _pendingOwner);
        _owner = _pendingOwner;
    }
```

The identified vulnerability in the *Ownable.sol* contract presents a significant security risk. Without resetting the *_pendingOwner* variable after ownership is renounced, there exists the potential for unexpected ownership regain. This deviation from the standard ownership renouncement process may be exploited, allowing a *_pendingOwner* to reclaim ownership through subsequent calls to the *updateOwner()* function.

Path: ./contracts/Ownable.sol : updateOwner();

Recommendation: It is recommended to reset the *_pendingOwner* variable to the zero address after ownership has been successfully transferred or renounced. This ensures that the pending owner status is cleared, preventing any further attempts to regain ownership through the *updateOwner()* function. The addition of this reset step enhances the security of the ownership management mechanism in the contract. This deviation from the standard

ownership renouncement process may be exploited, allowing the *_pendingOwner* to reclaim ownership through calls to the *updateOwner()* function.

Found in: 526a9f

Status: Fixed (Revised commit: 3002141)

Remediation: The *Ownable* library from OpenZeppelin was introduced.

Low

No low severity issues were found.

Informational

I01. Variable name shadowing issues in buckToken contract functions

This issue manifests in the current implementation, where variable names within a contract's functions overshadow variables of the same name within the contract scope itself. Such a situation can lead to confusion and unintended consequences during code execution.

The *owner* variable in the *buck.sol* contract is causing shadowing issues with the *owner()* function from *Ownable.sol*. This occurs in functions such as *transfer()*, *allowance()*, *approve()*, *increaseAllowance()*, *decreaseAllowance()*, *_approve()*, and *_spendAllowance()*.

Path: ./contracts/buck.sol : owner;

Recommendation: To mitigate the shadowing issue and enhance clarity in the code, consider renaming the owner variable in the *buck.sol* contract to a more

specific name that does not conflict with the *owner()* function inherited from *Ownable.sol*.

Found in: 526a9f

Status: Acknowledged (Revised commit: 3002141)

I02. Incomplete blacklisting check in buck.sol contract poses unauthorized fund reception vulnerability

The *buck.sol* contract utilizes a blacklisted mechanism to prevent transfers from addresses that are blacklisted in functions such as *_transfer()* and *transferFrom().* However, there is a vulnerability in the implementation as it fails to check whether the receiving address *to* is blacklisted. Consequently, a blacklisted address could receive funds in a transfer, without facing restrictions.

Path: ./contracts/buckToken.sol : transferFrom(), transfer();

Recommendation: To address this issue, it is recommended to implement a check on the blacklisted status of the to address in both the *_transfer()* and *transferFrom()* functions. This ensures that blacklisted addresses are prevented from receiving funds, maintaining the integrity of the blacklisting mechanism and bolstering the overall security of the token contract. The additional check can be performed before executing the transfer logic, and if the *to* address is blacklisted, the transaction should revert with an appropriate error message.

Found in: 526a9f

Status: Acknowledged (Revised commit: 3002141)

I03. Redundant check for zero address in ownership transfer in Ownable.sol

Within the *Ownable.sol* contract, there exists a redundancy in the *updateOwner()* check. Specifically, the condition *require(_msgSender() !=*

address(0), "Ownable: new owner is the zero address"); is present. However, this check is superfluous since the message sender *_msgSender()* can never be the zero address. Consequently, this condition will never be evaluated to *true*. Removing this redundant check can improve code clarity and decrease transaction Gas cost without compromising the security of the ownership transfer functionality.

Path: ./contracts/Ownable.sol : updateOwner();

Recommendation: It is recommended to remove the redundant check for the zero address in the ownership transfer logic.

Found in: 526a9f

Status: Fixed (Revised commit: 3002141)

Remediation: Redundant check for zero address was removed from *updateOwner().*

I04. Missing zero address check in blacklisting functions of buck.sol contract

Within the *buck.sol* contract, specifically in the functions *blacklistAddress()* and *removeBlacklistedAddress()*, there is an oversight in the form of a missing check for the zero address.

Path: ./contracts/BuckToken.sol : removeBlacklistedAddress(), blacklistAddress();

Recommendation: It is recommended to add a check to ensure that the provided user address in the *blacklistAddress()* and *removeBlacklistedAddress()* functions are not the zero address.

Found in: 526a9f

Hacken OU
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia

Status: Acknowledged (Revised commit: 3002141)

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.

Hacken OU
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia

Appendix 1. Severity Definitions

When auditing smart contracts Hacken is using a risk-based approach that considers the potential impact of any vulnerabilities and the likelihood of them being exploited. The matrix of impact and likelihood is a commonly used tool in risk management to help assess and prioritize risks.

The impact of a vulnerability refers to the potential harm that could result if it were to be exploited. For smart contracts, this could include the loss of funds or assets, unauthorized access or control, or reputational damage.

The likelihood of a vulnerability being exploited is determined by considering the likelihood of an attack occurring, the level of skill or resources required to exploit the vulnerability, and the presence of any mitigating controls that could reduce the likelihood of exploitation.

Risk Level	High Impact	Medium Impact	Low Impact
High Likelihood	Critical	High	Medium
Medium Likelihood	High	Medium	Low
Low Likelihood	Medium	Low	Low

Risk Levels

Critical: Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.

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.

Medium: Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.

Low: Major deviations from best practices or major Gas inefficiency. These issues will not have a significant impact on code execution, do not affect security score but can affect code quality score.

Impact Levels

High Impact: Risks that have a high impact are associated with financial losses, reputational damage, or major alterations to contract state. High impact issues typically involve invalid calculations, denial of service, token supply manipulation, and data consistency, but are not limited to those categories.

Medium Impact: Risks that have a medium impact could result in financial losses, reputational damage, or minor contract state manipulation. These risks can also be associated with undocumented behavior or violations of requirements.

Low Impact: Risks that have a low impact cannot lead to financial losses or state manipulation. These risks are typically related to unscalable functionality, contradictions, inconsistent data, or major violations of best practices.

Likelihood Levels

High Likelihood: Risks that have a high likelihood are those that are expected to occur frequently or are very likely to occur. These risks could be the result of known vulnerabilities or weaknesses in the contract, or could be the result of external factors such as attacks or exploits targeting similar contracts.

Medium Likelihood: Risks that have a medium likelihood are those that are possible but not as likely to occur as those in the high likelihood category. These risks could be the result of less severe vulnerabilities or weaknesses in the contract, or could be the result of less targeted attacks or exploits.

Low Likelihood: Risks that have a low likelihood are those that are unlikely to occur, but still possible. These risks could be the result of very specific or complex vulnerabilities or weaknesses in the contract, or could be the result of highly targeted attacks or exploits.

Informational

Informational issues are mostly connected to violations of best practices, typos in code, violations of code style, and dead or redundant code.

Informational issues are not affecting the score, but addressing them will be beneficial for the project.

Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

Scope details

Repository	https://github.com/Coinbuck/Contracts/tree/development
Commit	526a9f

Contracts in Scope

./contract/token/IERC20Metadata.sol;
./contract/token/IERC20.sol;
./contract/utils/Context.sol;
./contract/access/Ownable.sol;
./contract/buck.sol.sol

Second scope details

Repository	https://github.com/Coinbuck/Contracts/tree/development
Commit	3002141

Hacken OU
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia

Contracts in Second Scope

./contract/buck.sol.sol

Thidr scope details

Repository	https://github.com/Coinbuck/Contracts/tree/development
Commit	f8ba87

Contracts in Third Scope

./contract/buck.sol.sol