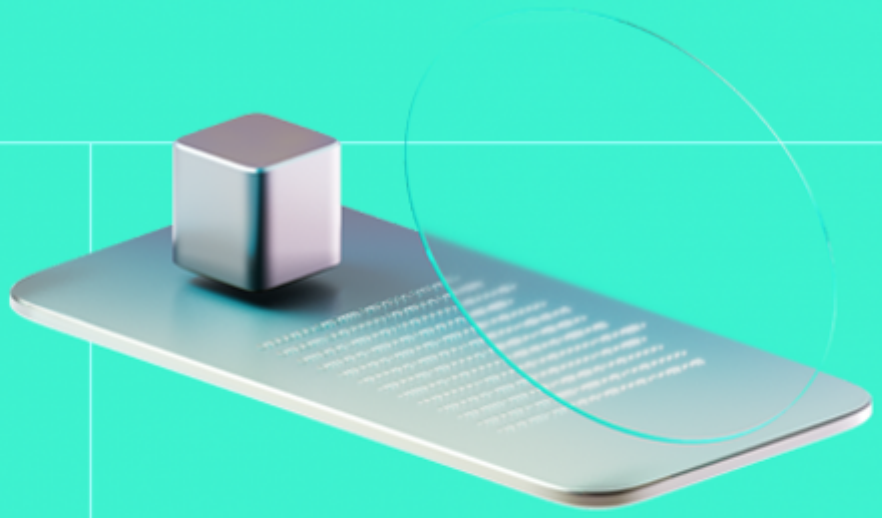
HACKEN

Smart Contract Code Review And Security Analysis Report



Customer: Coins & Skins

Date: 29/01/2024

We express our gratitude to the Coins & Skins team for the collaborative engagement that enabled the execution of this Smart Contract Security Assessment.

COINS & SKINS is an ERC-20 tokens.

Platform: Ethereum

Language: Solidity

Tags: ERC-20

Timeline: 23/01/2024 - 25/01/2024

Methodology: https://hackenio.cc/sc_methodology

Review Scope

Repository	https://github.com/holdex/SKINS-token
Commit	1df37236811ccd60c645bec1ec2a5b86c66e813d

Audit Summary

10/10	8/10	100%	10/10
Security Score	Code quality score	Test coverage	Documentation quality score

Total 9.6/10

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

0	0	0	0
Total Findings	Resolved	Accepted	Mitigated

Findings by severity

Critical	0
High	0
Medium	0
Low	0

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 Coins & Skins
Audited By	Maksym Fedorenko
Approved By	Ataberk Yavuzer
Website	https://coinsandskins.com/
Changelog	26/01/2024 - Preliminary Report
	29/01/2024 - Final Report

Table of Contents

System Overview

Coins & Skins is an ERC-20 token contract.

It has the following attributes:

- Deployment target chain: Ethereum.
- Contract Type: ERC-20.
- Total token supply: 800,000,000.
- Burnable: no.
- Mintable: no (fixed supply).

The contract includes the next libraries and contracts:

Value — is an abstract contract. It contains a modifier named nonZA that ensures a function is not called with the zero address as the sender. It declares a custom error NonZeroAddress to handle such cases.

SafeMath — is a custom abstract contract, which provides functions for performing arithmetic operations with safety checks to prevent overflows and underflows.

Votes — is the contract with the votes delegation logic (out of the audit scope).

Nonces — is a custom contract designed to manage and validate nonces for signed messages related to the vote delegation.

Privileged roles

- There is no explicit concept of an owner.
- The deployer, being the entity that deploys the contract to the blockchain, has control over the contract deployment process but does not have special privileges or control over the contract's behavior after deployment unless explicitly specified within the contract's functions or state variables.

Executive Summary

This report presents an in-depth analysis and scoring of the customer's smart contract project. Detailed scoring criteria can be referenced in the scoring methodology.

Documentation quality

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

- Functional token requirements are clear.
- Technical token description is provided within the Readme file.
- NatSpec covers the code and is detailed.

Code quality

The total Code Quality score is **8** out of **10**.

- The code has redundant code blocks with unused functions and variables.
- The code mixes `uint` and `uint256` type declarations.

Test coverage

Code coverage of the project is **100%**.

Security score

Upon auditing, the code was found to contain **0** critical, **0** high, **0** medium, and **0** low severity issues, leading to a security score of **10** out of **10**.

All identified issues are detailed in the “Findings” section of this report.

Summary

The comprehensive audit of the customer's smart contract yields an overall score of **9.6**. This score reflects the combined evaluation of documentation, code quality, test coverage, and security aspects of the project.

Risks

- The functionality related to the `Votes.sol` smart contract is out of scope and should not be utilized in production.

Findings

Observation Details

F-2024-0596 - Redundant Code Block - Info

Description: The contract defines an error condition named `AmountOverflow()`, presumably intended for handling arithmetic overflow situations. However, this error condition is not utilized anywhere else in the contract's logic or functions.

Redundant parts of the code create excessive gas costs.

Assets:

- SafeMath.sol [https://github.com/hknio/SKINS-token-1ebcc179ca5f349e1df37f6dd5a/blob/main/contracts/utils/SafeMath.sol]

Status: Accepted

Recommendations

Recommendation: Remove redundant code blocks in order to consume less gas.

Remediation: The Coins & Skins team acknowledged the finding.

F-2024-0628 - EIP-20 Standard Violation - Info

Description:

The codebase implements the total supply public constant `uint96 public constant totalSupply = 800_000_000e18;` . The compiler automatically creates a getter for such constant, consecuently the getter has a `uint96` return value type. However this contradicts the EIP-20 standard, which states that the `totalSupply()` getter should return the value of the `uint256` type: `function totalSupply() public view returns (uint256)`

This might lead to the issues during the integration of the project with the third party systems.

Assets:

- Skins.sol [https://github.com/hknio/SKINS-token-1ebcc179ca5f349e1df37f6dd5a/blob/main/contracts/Skins.sol]

Status: Accepted

Recommendations

Recommendation:

Modify the `totalSupply` constant type to `uint256` to align with the specifications outlined in the EIP-20. Ensure that any changes made do not compromise the overall functionality and security of the smart contract.

Remediation: The Coins & Skins team acknowledged the finding.

F-2024-0629 - The Code Mixes `uint` And `uint256` Type Declaration - Info

Description: Most of the code utilize `uint256` for type declarations, yet the `allowance` function uses `uint` for the return value. This reduces code readability and could impact future supportability.

```
function allowance(
address account,
address spender
) external view returns (uint) {
return _allowances[account][spender];
}
```

Assets:

- Skins.sol [https://github.com/hknio/SKINS-token-1ebcc179ca5f349e1df37f6dd5a/blob/main/contracts/Skins.sol]

Status: Accepted

Recommendations

Recommendation: Substitute `uint` with `uint256`. Ensure correct conversion of `uint96` allowance into `uint256` type.

Remediation: The Coins & Skins team acknowledged the finding.

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.

Appendix 1. Severity Definitions

When auditing smart contracts, Hacken is using a risk-based approach that considers **Likelihood**, **Impact**, **Exploitability** and **Complexity** metrics to evaluate findings and score severities.

Reference on how risk scoring is done is available through the repository in our Github organization:

hknio/severity-formula

Severity	Description
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.

Appendix 2. Scope

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

Scope Details

Repository	https://github.com/holdex/SKINS-token
Commit	1df3723
Whitepaper	https://github.com/holdex/SKINS-token/blob/main/README.md
Requirements	NatSpec
Technical Requirements	https://github.com/holdex/SKINS-token/blob/main/README.md

Contracts in Scope

./contracts/Skins.sol

./contracts/utils/Nonces.sol

./contracts/utils/Value.sol

./contracts/utils/SafeMath.sol