HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Dexola
Date: 21 Nov, 2023

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 Dexola
Approved By	David Camps Novi | Lead Solidity SC Auditor at Hacken OÜ Paul Fomichov | Approver SC Auditor at Hacken OÜ
Tags	ERC721 token; ERC2981 royalties;
Platform	EVM
Language	Solidity
Methodology	Link
Website	https://dexola.com/
Changelog	10.11.2023 - Initial Review 21.11.2023 - Second Review

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by Dexola (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

The Separate Assault CryptBrigade is an ERC721 NFT single contract with the following features:

- Maximum supply of 10000 tokens.
- Mint upon purchase at 0.05 ETH per token.
- ERC2981 royalties.

Privileged roles

- **Owner**: contract owner who can activate the sales, withdraw native tokens from the contract, update base and token URI, update base and token royalties.

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.
- Technical description is provided.

Code quality

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

- The development environment is configured.

Test coverage

Code coverage of the project is **100%** (branch coverage).

- Deployment and basic user interactions are covered with tests.

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

1
2
3
4
5
6
7
8
9
10
The final score

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
10 November 2023	2	0	0	0
21 November 2023	0	0	0	0

Risks

- Royalties are introduced via ERC2981, but they are not enforced on-chain. Therefore, their off-chain management cannot be reviewed in this audit.
- ERC721 NFTs can be transferred without the payment of royalties since they are implemented off-chain.

Findings

■■■■ Critical

No critical severity issues were found.

■■■ High

No high severity issues were found.

■■ Medium

No medium severity issues were found.

■ Low

L01. Redundant check consumes extra Gas

Impact	Low
Likelihood	Medium

In the *constructor()*, a check is implemented to make sure the *feeRecipient_* address is not the *0x0* address. However, said check is already implemented into ERC2981's *_setDefaultRoyalty()*, spending unnecessary extra Gas.

Path: ./contracts/SACB.sol: constructor().

Recommendation: it is recommended to remove the redundant *0x0* check.

Found in: 267dbd3

Status: Fixed (Revised commit: 552155d)

Resolution: The redundant zero address check was removed from the *constructor()*.

L02. Missing URI length check

Impact	Medium
Likelihood	Low

The methods *constructor()*, *updateBaseURI()*, *updateURI()*, *updateURIBatch()* lack an empty *URI* string check, which may lead to unexpected behavior.

Paths: ./contracts/SACB.sol: *tokenURI().*

Recommendation: it is recommended to introduce an empty *URI* check.

Found in: 267dbd3

Status: Fixed (Revised commit: 552155d)

Resolution: Empty *URI* checks were implemented in *updateURI()* and *updateURIBatch()*. In case of base *URI* in *constructor*() and *updateBaseURI()*, empty checks were not implemented since base *URI* can be empty.

Informational

I01. Unfollowed explicit uint best practice

It is a recommended best practice to explicitly declare the size of *uint256* variables instead of *uint*.

Path: ./contracts/SACB.sol: MAX_SUPPLY; PRICE; _withdraw() → balance.

Recommendation: consider declaring explicit *uint256* instead of *uint*.

Found in: 267dbd3

Status: Fixed (Revised commit: 552155d)

Resolution: Explicit *uint256* was implemented in reported variables.

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 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 won't have a significant impact on code execution, don't 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:

Initial review scope

Repository	https://github.com/dexolacom/comebackalive-smart-contracts
Commit	267dbd3
Whitepaper	Not provided
Requirements	./README.md
Technical Requirements	./README.md
Contracts	File: ./contracts/SACB.sol

Second review scope

Repository	https://github.com/dexolacom/comebackalive-smart-contracts
Commit	552155d
Whitepaper	Not provided
Requirements	./README.md
Technical Requirements	./README.md
Contracts	File: ./contracts/SACB.sol