HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Embr Holdings Limited
Date: July 04th, 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 Embr Holdings Limited
Approved By	Evgeniy Bezuglyi | SC Department Head at Hacken OU
Type	DEX Router
Platform	Not Specified
Network	Not Specified
Language	Solidity
Methods	Manual Review, Automated Review, Architecture review
Website	https://joinembr.com
Timeline	13.05.2022 - 04.07.2022
Changelog	16.05.2022 - Initial Review 24.05.2022 - Second Review 04.07.2022 - Third 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 Embr Holdings Limited (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 Checkout Contract in the repository:

Initial review scope

Repository: https://github.com/teamembr/router-contract/blob/main/contracts/Checkout.sol
Commit:
588e669
Technical Documentation:
Type: Litepaper (no functional requirement provided)
Link: https://coda.io/@joinembr/embr-litepaper

Integration and Unit Tests: No
Contracts:
File: ./contracts/Checkout.sol
SHA3: bc8a63388ccf4b3a8d2610b7228db043aab2fb0ebf415e206732b97d84cb34df

Second review scope

Repository: https://github.com/teamembr/router-contract/blob/main/contracts/Checkout.sol
Commit:
fc00c60
Technical Documentation:
Type: Litepaper (no functional requirement provided)
Link: https://coda.io/@joinembr/embr-litepaper
Integration and Unit Tests: No
Contracts:
File: ./contracts/Checkout.sol
SHA3: f8ce1ade955e51649b038f6ae25676abccf23b3b8293071aa4a752e94d462498

Third review scope

Repository: https://github.com/teamembr/router-contract/blob/main/contracts/Checkout.sol
Commit:
a00770c
Technical Documentation:
Type: Litepaper (no functional requirement provided)
Link: https://coda.io/@joinembr/embr-litepaper

Type: Technical Requirements
Link: https://github.com/teamembr/router-contract/blob/main/REQUIREMENTS.md
Integration and Unit Tests: Yes
Contracts:
File: ./contracts/Checkout.sol
SHA3: 073f3523da0d8543c84e4235781270378d414f723b5820d339a6d6bd6cf9ae60

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 Customer provided both functional and technical requirements. The total Documentation Quality score is 10 out of **10**.

Code quality

The total CodeQuality score is **8** out of **10**. Code follows official language style guides. Some of the positive and negative tests are missing.

Architecture quality

The architecture quality score is **10** out of **10**. Clean and clear architecture, well structured and explained development environment.

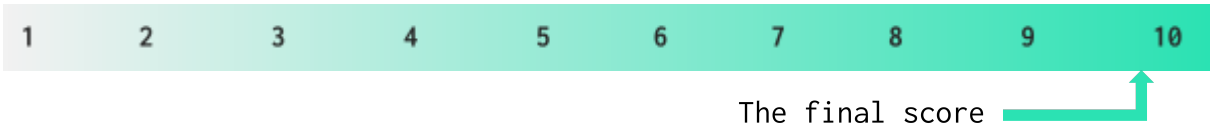
Security score

As a result, the code contains **0** 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: **9.8**.



Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

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.	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.	Passed
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 destroyed until 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
Uninitialized Storage Pointer	SWC-109	Storage type should be set explicitly if the compiler version is < 0.5.0.	Not Relevant
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 through tx.origin	SWC-115	tx.origin should not be used for authorization.	Passed
Block values as a proxy for time	SWC-116	Block numbers should not be used for time calculations.	Not Relevant
Signature Unique Id	SWC-117 SWC-121 SWC-122	Signed messages should always have a unique id. A transaction hash should not be used as a unique id.	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.	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.	Not Relevant
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.	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.	Not Relevant
Repository Consistency	**Custom**	The repository should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed
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

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

System Overview

Embr Checkout is a Decentralized Exchange Router system. The audit scope verifies only the **Checkout** contract. Though, the security of the Decentralized Exchange platforms with which the contract interacts is not included in this scope.

- Checkout - a contract that forwards the swap orders to the allowed Decentralized Exchange platforms with a **1%** fee deduction.

Privileged roles

- The Checkout contract has one privileged role named “owner” which has the following privileges:
  - Can set supported DEX addresses
  - Can specify DEX addresses that have a different interface
  - Can change fee rate
  - Can withdraw Ethers from the contract

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Findings

Critical

No critical severity issues were found.

High

No high severity issues were found.

Medium

1. **Missing edge case check for parameter**

   According to the contract logic, “fee” variable cannot be bigger than 10000. However, there is no check in the "setFee" function for this case.

   This can cause errors during fee calculations.

   Contract: Checkout.sol

   Function: setFee

   Recommendation: Implement edge case check.

   Status: Fixed (Revised commit: fc00c60)

2. **Impractical code block**

   "swapExactETHForTokens" function checks if the provided "exchangeKey" is "traderjoe" because Traderjoe's router contract function names use AVAX instead of ETH. However, some other DEX platforms on Avalanche Blockchain use this pattern, but it is impossible to interact with these DEX platforms using the current version of the contract.

   Contract: Checkout.sol

   Function: swapExactETHForTokens

   Recommendation: Check whether the mentioned problem affects the functionality of the system. If so, revise the function.

   Status: Fixed (Revised commit: fc00c60)

3. **Usage of Gas-Inefficient methods**

   In "swapExactETHForTokens" function, string comparison has been preferred to check provided key, and string mapping has been preferred to reach DEX addresses. However, it is Gas-inefficient to use strings for mappings and comparisons in Solidity.

   This issue leads to high Gas usage.

   Contract: Checkout.sol

   Function: swapExactETHForTokens

Recommendation: Take DEX address as input parameter and use "address to bool" mapping to check is supported or not. Use "address to bool" mapping instead of string comparison.

Status: Fixed (Revised commit: fc00c60)

Low

1. Unlocked pragma

Unlocked pragmas may cause the contract to be deployed with a different Solidity version from the tested.

This can lead to encountering undiscovered bugs.

Contract: Checkout.sol

Function: -

Recommendation: Lock pragma to a specific compiler version.

Status: Fixed (Revised commit: fc00c60)

2. Using SafeMath

SafeMath is generally not needed after Solidity version 0.8.

Contract: Checkout.sol

Function: -

Recommendation: Remove SafeMath.

Status: Fixed (Revised commit: fc00c60)

3. Missing zero address validation

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

This can lead to unwanted external calls to 0x0.

Contract: Checkout.sol

Functions: constructor, swapExactETHForTokens, setDexRouter, setTreasuryAddress, withdraw

Recommendation: Implement zero address checks.

Status: Fixed (Revised commit: fc00c60)

4. Missing zero amount validation

Message value is being used without checking against the possibility of 0.

This can lead to unnecessary transfer calls.

Contract: Checkout.sol

Function: swapExactETHForTokens

Recommendation: Implement zero amount check

Status: Fixed (Revised commit: fc00c60)

5. **Unused libraries**

   "IERC20", "SafeERC20" and "Address" libraries have no implementation on the contract.

   Keeping unused libraries increases Gas costs during deployment.

   Contract: Checkout.sol

   Function: -

   Recommendation: Remove unused libraries.

   Status: Fixed (Revised commit: a00770c)

6. **Redundant variable**

   "treasuryAddress" variable has no effect on the contract logic.

   Keeping redundant variables increases Gas costs during deployment.

   Contract: Checkout.sol

   Function: -

   Recommendation: Remove redundant variables.

   Status: Fixed (Revised commit: fc00c60)

7. **Missing event emitting**

   Events for critical state changes should be emitted for tracking things off-chain.

   Contract: Checkout.sol

   Functions: setDexRouter, setFee, withdraw

   Recommendation: Create and emit related events.

   Status: Fixed (Revised commit: fc00c60)

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

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 audit makes no statements or warranties on the security of the code. It also cannot be considered a 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.

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 audit cannot guarantee the explicit security of the audited smart contracts.