HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Embr
Date: 26 Jul, 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 Embr
Approved By	Paul Fomichov | Lead Solidity SC Auditor at Hacken OU
Tags	Signatures; Proxy
Platform	EVM
Language	Solidity
Methodology	Link
Website	https://www.embr.org/
Changelog	1.06.2023 - Initial Review 27.06.2023 - Second Review 26.07.2023 - 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 (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 Vault contract is a Solidity smart contract that implements functionality for token swapping tokens, it forwards calls to 1inch v5: an Aggregation Router contract. The contract incorporates essential features from the OpenZeppelin libraries, including access control through Ownable, token handling using SafeERC20, and signature verification through EIP712. Notably, the contract owner has the authority to manage the addresses eligible for signature verification and has the ability to withdraw ERC20 tokens unintentionally sent to the contract.

Privileged roles

- The Vault contract provides functionality for the owner to manage signatories for signature verification and grants them the capability to withdraw ERC-20 tokens from the contract. The addition or removal of signatories is at the discretion of the contract owner, allowing for flexibility in the signatory management process.

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 scoring methodology.

Documentation quality

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

- Functional requirements are detailed.
- Technical description is robust.
- NatSpec is sufficient.

Code quality

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

- No Solidity Style Guide violations.
- No best practice violations.

Security score

As a result of the audit, 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: **10**. The system users should acknowledge all the risks summed up in the risks section of the report.

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
30 May 2023	5	1	1	1
27 June 2023	2	2	1	0
26 July 2023	0	0	0	0

Risks

- *_data* parameter for transactions in the *swapUsdcToTargetToken()* and *swap()* functions is generated via 1inch router API. If this parameter is supplied through the backend system and is compromised by an attacker, there exists a potential security vulnerability. The attacker could manipulate one of the parameters utilized in generating the *_data* argument, specifically the destReceiver address, allowing them to redirect the exchanged tokens to their own address.

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

Checked Items

We have audited the Customers' smart contracts for commonly known and specific vulnerabilities. Here are some items considered:

Item	Description	Status	Related Issues
Default Visibility	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed	
Integer Overflow and Underflow	If unchecked math is used, all math operations should be safe from overflows and underflows.	Not Relevant	
Outdated Compiler Version	It is recommended to use a recent version of the Solidity compiler.	Passed	
Floating Pragma	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Passed	
Unchecked Call Return Value	The return value of a message call should be checked.	Passed	
Access Control & Authorization	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	The contract should not be self-destructible while it has funds belonging to users.	Not Relevant	
Check-Effect-Interaction	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed	
Assert Violation	Properly functioning code should never reach a failing assert statement.	Passed	
Deprecated Solidity Functions	Deprecated built-in functions should never be used.	Passed	
Delegatecall to Untrusted Callee	Delegatecalls should only be allowed to trusted addresses.	Not Relevant	
DoS (Denial of Service)	Execution of the code should never be blocked by a specific contract state unless required.	Passed	

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

Race Conditions	Race Conditions and Transactions Order Dependency should not be possible.	Passed	
Authorization through tx.origin	tx.origin should not be used for authorization.	Not Relevant	
Block values as a proxy for time	Block numbers should not be used for time calculations.	Not Relevant	
Signature Unique Id	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifiers should always be used. All parameters from the signature should be used in signer recovery. EIP-712 should be followed during a signer verification.	Passed	
Shadowing State Variable	State variables should not be shadowed.	Passed	
Weak Sources of Randomness	Random values should never be generated from Chain Attributes or be predictable.	Not Relevant	
Incorrect Inheritance Order	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	All external calls should be performed only to trusted addresses.	Passed	
Presence of Unused Variables	The code should not contain unused variables if this is not justified by design.	Passed	
EIP Standards Violation	EIP standards should not be violated.	Passed	
Assets Integrity	Funds are protected and cannot be withdrawn without proper permissions or be locked on the contract.	Passed	
User Balances Manipulation	Contract owners or any other third party should not be able to access funds belonging to users.	Not Relevant	
Data Consistency	Smart contract data should be consistent all over the data flow.	Passed	

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

Flashloan Attack	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. Contracts shouldn't rely on values that can be changed in the same transaction.	Not Relevant	
Token Supply Manipulation	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	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.	Not Relevant	
Style Guide Violation	Style guides and best practices should be followed.	Passed	
Requirements Compliance	The code should be compliant with the requirements provided by the Customer.	Passed	
Environment Consistency	The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed	
Secure Oracles Usage	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	The code should be covered with unit tests. Test coverage should be sufficient, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Failed	
Stable Imports	The code should not reference draft contracts, which may be changed in the future.	Passed	

Findings

■■■■ Critical

C01. Signed Message Replay Attack; Irrevocable Signed Message

Impact	High
Likelihood	High

The protocol's pool and staking contracts, which inherit from VerifySignatureSystem, are not utilizing EIP712 for signatures and deadlines. The current implementation of signed messages lacks mechanisms to prevent replay attacks and does not offer a mechanism for message expiration.

Without a standard such as EIP712, signed messages can potentially be reused on other chains (replay attacks).

This can lead to unauthorized execution of smart contract functions, manipulation of the contract's state and data, or compromise of the integrity and security of the contract's operations.

The signature revoke issue arises when a contract lacks functionality that allows revoking a signed message.

This can result in the execution of a signed transaction when it is not desired by a signer.

Path: ./contracts/Vault.sol : verifySignature()

Recommendation: Implement EIP712 for signatures in the protocol. EIP712 offers a standard way to structure data and generate signatures, which would significantly enhance the security of the contract by protecting it from replay attacks. Consider adding an expiration timestamp (deadline) to the signed messages to ensure that they cannot be used indefinitely, further increasing the robustness of the protocol. Implement a mechanism to prevent re-use of the already used message hash.

Found in: 2675537

Status: Fixed (Revised commit: f6b8542)

■■■ High

H01. Missing Validation

Impact	High
Likelihood	Medium

Insufficient validation is observed within the *swapUsdcToTargetToken()* and *swap()* functions, wherein the provided swap arguments (*amount*, *srcToken*, and *dstToken*) lack validation

against the corresponding arguments derived from the off-chain creation of the *_data* argument.

This can lead to unexpected contract behavior.

Path: ./contracts/Vault.sol : swap(), swapUsdcToTargetToken()

Recommendation: Verify if values in *_data* are the same as *amount*, *srcToken* and *dstToken* function parameters or create function call data inside function.

Found in: 2675537

Status: Mitigated(*_data* parameter is generated via 1inch API and validation check is done in the backend system.)

Medium

M01. Unchecked Transfer or Approve

Impact	Medium
Likelihood	Medium

It is considered following best practices to avoid unclear situations and prevent common attack vectors.

The functions do not use the SafeERC20 library for checking the result of ERC20 token transfer and approvals. Tokens may not follow the ERC20 standard and return false in case of transfer failure or not returning any value at all.

This may lead to denial of service vulnerabilities when interacting with non-standard tokens.

Path: ./contracts/Vault.sol : swap(), swapUsdcToTargetToken()

Recommendation: Use the SafeERC20 library to interact with tokens safely.

Found in: 2675537

Status: Fixed (Revised commit: d266126)

M02. Copy Of Well Known Contract

Impact	Low
Likelihood	High

Well-known contracts from projects like *EIP712* from OpenZeppelin should be imported directly from source as the projects are in development and may update the contracts in future.

The system uses a copy of OpenZeppelin *EIP712*.

Path: ./contracts/Vault.sol

Recommendation: Import the contract directly from source, avoid modifying them.

Found in: f6b8542

Status: Fixed (Revised commit: d266126)

Low

L01. Missing Zero Address Validation

Impact	Low
Likelihood	Medium

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

This can lead to unwanted external calls to 0x0.

Path: ./contracts/Vault.sol : constructor(), emergencyWithdraw()

Recommendation: Implement zero address checks.

Found in: 2675537

Status: Fixed (Revised commit: f6b8542)

L02. State Variables Can Be Declared Immutable

Impact	Low
Likelihood	Low

Variable`s *USDC_ADDRESS* and *AGGREGATION_ROUTER_V5* values are set in the constructor. This variable can be declared immutable.

This will lower the Gas taxes.

Path: ./contracts/Vault.sol : USDC_ADDRESS, AGGREGATION_ROUTER_V5

Recommendation: Declare mentioned variables as immutable.

Found in: 2675537

Status: Fixed (Revised commit: f6b8542)

L03. Missing Events

Impact	Low
Likelihood	Low

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

Path: ./contracts/Vault.sol : removeSignatory(), addSignatory(), emergencyWithdraw(), swapUsdcToTargetToken()

Recommendation: Create and emit related events.

Found in: 2675537

Status: Fixed (Revised commit: d266126)

L04. Docs Mismatch

Impact	Low
Likelihood	Low

The project should be consistent and contain no self contradictions.

According to documentation, the *swapUsdcToTargetToken()* function is intended to be exclusively accessible to the admin.

> *Have admin function which swap USDC token to the target token.*

However, in the implementation, there is a missing verification mechanism to ascertain whether the invoking entity (*msg.sender*) possesses the necessary administrative privileges.

This may lead to unexpected contract behavior.

Path: ./contracts/Vault.sol

Recommendation: Fix the mismatch.

Found in: 2675537

Status: Fixed (Revised commit: d266126)

L05. Unused Return Value

Impact	Low
Likelihood	Low

The return value *_returnData* is stored during function execution but never used.

This makes code harder to read and more expensive.

Path: ./contracts/Vault.sol: swapUsdcToTargetToken(), swap()

Recommendation: Remove unused variable.

Found in: 2675537

Status: Fixed (Revised commit: f6b8542)

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

Informational

I01. Style Guide Violation

Contract readability and code quality are influenced significantly by adherence to established style guidelines. In Solidity programming, there exist certain norms for code arrangement and ordering. These guidelines help to maintain a consistent structure across different contracts, libraries, or interfaces, making it easier for developers and auditors to understand and interact with the code.

The suggested order of elements within each contract, library, or interface is as follows:

1. Type declarations
2. State variables
3. Events
4. Modifiers
5. Functions

Functions should be ordered and grouped by their visibility as follows:

1. constructor
2. receive function (if exists)
3. fallback function (if exists)
4. external
5. public
6. internal
7. private

Within each grouping, *view* and *pure* functions should be placed at the end.

Furthermore, following the Solidity naming convention and adding NatSpec annotations for all functions is strongly recommended. These measures aid in the comprehension of code and enhance overall code quality.

Path: ./contracts/Vault.sol

Recommendation: Reorder the functions to adhere to the official Solidity Style Guide. This enhances readability and maintainability of the code, facilitating seamless interaction with the contracts.

Found in: 2675537

Status: Fixed (Revised commit: d266126)

I02. No Messages In Revert

Some *revert()* statements are missing revert messages.

This makes the code harder to test and debug.

Path: ./contracts/Vault.sol: swapUsdcToTargetToken(), swap()

Recommendation: Add revert message to the *revert()* statement in *swapUsdcToTargetToken()*. Provide a more detailed message to the *revert()* statement in the *swap() function*.

Found in: 2675537

Status: Fixed (Revised commit: f6b8542)

I03. Style Guide Violation - Naming Conventions

The project should follow the official code style guidelines.

Local and State Variable Names should be named with mixedCase.

Path: ./contracts/Vault.sol : DOMAIN_SEPARATOR

Recommendation: The official Solidity style guidelines should be followed.

Found in: f6b8542

Status: Fixed (Revised commit: d266126)

I04. Redundant Import

The contract imports OpenZeppelin's *IERC20* but it is already part of *SafeERC20*.

Path: ./contracts/Vault.sol

Recommendation: Remove redundant inheritance to save Gas on deployment and increase the code quality.

Found in: f6b8542

Status: Fixed (Revised commit: d266126)

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

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.

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

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/teamembr/router-contract
Commit	26755371c5e6e2e5602cc5c21a27224f644b3185
Whitepaper	https://whitepaper.mattr.one/
Requirements	Link
Technical Requirements	Link
Contracts	File: contracts/Vault.sol SHA3: 7ebf279a04aa330093c42042bd24c880f3c31e008fc880c774925e300f43b3fe

Second review scope

Repository	https://github.com/teamembr/router-contract
Commit	f6b854287cd0035b236db619ca8802f8dcf0de34
Whitepaper	https://whitepaper.mattr.one/
Requirements	Link
Technical Requirements	Link
Contracts	File: contracts/Vault.sol SHA3: d585d351d1956cdf857763db32a94ba33ab4d8c40108e8ba0aab335b62fa246e

Third review scope

Repository	https://github.com/teamembr/router-contract
Commit	d266126002ac9c4d39767332706ad692ff45c509
Whitepaper	https://whitepaper.mattr.one/
Requirements	Link
Technical Requirements	Link
Contracts	File: contracts/Vault.sol SHA3: 733733d87a125d0a36562b112639622875e665f16176e24e3f51f0e1f96c7ff2