HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Dexalot
Date: 03 November, 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 Dexalot
Approved By	Viktor Lavrenenko | SC Auditor at Hacken OÜ David Camps Novi | SC Audits Lead at Hacken OÜ Paul Fomichov | SC Audits Approver at Hacken OÜ
Tags	DEX, Signatures;
Platform	Avalanche
Language	Solidity
Methodology	Link
Website	https://dexalot.com/
Changelog	25.04.2023 - Initial Review 16.05.2023 - Second Review 22.05.2023 - Third Review 05.09.2023 - Fourth Review 15.09.2023 - Fifth Review 27.10.2023 - Sixth Review 03.11.2023 - Seventh 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 Dexalot (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 scope of this audit consists of an upgradeable contract that handles swapping of any two assets based on a signed quote that is generated through an off-chain REST API. The swapping details, such as the amounts and receivers, are determined by the quote generated by the REST API.
The latest version of the system implements functionality to allow signature verifications from non-EOA smart contracts.

The files in the scope:

- **MainnetRFQ.sol** - The contract that handles the signature verified swapping.

Privileged roles

- swapSigner: creates signature.
- rebalancer: rebalances inventory of the smart contract, updates quote expiry and quote maker amount.
- default admin: manages swapSigner and rebalancer addresses. Sets trusted contracts, changes the admin, and can pause/unpause the contract, set slippage tolerance.

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 present.
- Technical specifications, including NatSpec, are detailed.
- Description of the development environment is sufficient.

Code quality

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

- The development environment is configured.
- Best practices are followed.

Test coverage

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

- Deployment and basic user interactions are covered with tests.
- Negative cases are covered.

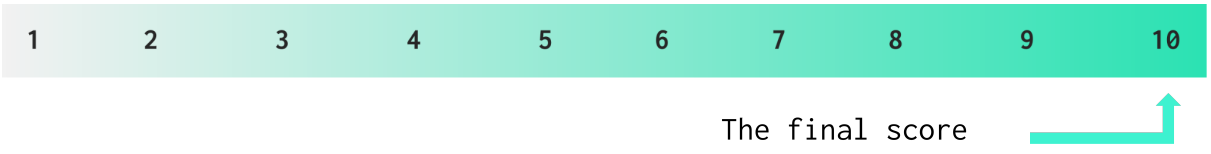
Security score

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.



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

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
25 April 2023	5	0	1	0
16 May 2023	1	0	0	0
22 May 2023	0	0	0	0
05 September 2023	1	1	1	0
15 September 2023	0	0	0	0
27 October 2023	0	0	0	0
03 November 2023	0	0	0	0

Risks

- The **off-chain REST API used to get a signed quote** that also **determines the swap rate** of the **assets** is **out of this audit scope** and its **security can not be guaranteed**.
- Block.timestamp values are used for swaps; hence, it creates a risk of manipulation.
- Missing *_disableInitilizers()* call in the constructor creates a risk of the implementation contract being directly initialized.
- There is no token whitelisting, and therefore users can introduce any token compliant with ERC-20. Fee-on-transfer tokens can be also used, which may cause unexpected behavior if they are not correctly handled off-chain.
- A big part of the project is handled off-chain by the RFQ API, and cannot be audited by the Hacken team.
    - Creation and management of orders in the swaps.
    - Management of signatures introduced by the system.

  As a consequence, the users need to deeply rely on the protocol owners to act correctly due to the high centralization.
- The contract is upgradeable, which means the protocol owners can change the logic of the contract without previous notice.
- In times of high volatility, the API may adjust the quoted price. The price will never be lower than slippageTolerance, which represents a percentage of the original quoted price. To check if the quoted price has been affected by slippage, the SlippageApplied event should be monitored. The expiry of the quote may also be adjusted during times of high volatility.
- When partialSwap is called, an order is not wholly filled.

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.	Passed	
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.	Passed	
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.	Passed	
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.	Passed	
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.	Passed	
Stable Imports	The code should not reference draft contracts, which may be changed in the future.	Passed	

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

Findings

■■■■ Critical

No critical severity issues were found.

■■■ High

H01. Upgradeability Issues

Impact	Medium
Likelihood	High

The contract is upgradable but does not follow the upgradability best practices by not adding a gap in the contract storage.

This may lead to contract storage layout corruption during an upgrade.

The contract inherits EIP712Upgradeable that contains a __gap variable, but it is a best practice to create a new __gap variable that will be more accessible due to variables order.

Path: ./contracts/MainnetRFQ.sol

Recommendation: Add a gap to the contract storage to allow future upgradability.

Found in: f8881f9

Status: Fixed

(Revised commit: 4d650f9)

H02. Unsafe Approval

Impact	High
Likelihood	High

The contract MainnetRFQ uses the *approve()* function inside of the *_executeSwap()*, which does not update the allowance, but replaces it.

This creates a problem in a situation, when a taker, which is a smart-contract, makes several swaps, and does not withdraw the previous approval.

Path: ./contracts/MainnetRFQ.sol: *_executeSwap()*

Proof of Concept: **Dexalot PoC**

Recommendation: Transfer tokens to the contract in a direct way or use *safeIncreaseAllowance()* method from SafeERC20Upgradeable library.

Found in: bc4b5dd

Status: Fixed (Revised commit: f13e089)

Medium

M01. EIP Standard Violation: Missing Value Check

Impact	High
Likelihood	Low

According to the EIP-1271 implementation, the *s* value in the signature verification process should be checked against an upper value. The function *_recoverSigner()* does not implement an upper bound check for the variable *s*.

Path: ./contracts/MainnetRFQ.sol: *_recoverSigner()*

Recommendation: Follow the EIP-1271 standard and implement a check for the value *s*.

Found in: bc4b5dd

Status: Fixed (Revised commit: f13e089)

Low

L01. Missing Zero Address Validation

Impact	Low
Likelihood	Medium

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

This can lead to unwanted external calls to 0x0.

Path: ./contracts/MainnetRFQ.sol: *initialize()*, *addAdmin()*, *addTrustedContract()*

Recommendation: Implement zero address checks.

Found in: f8881f9

Status: Fixed

(Revised commit: 4d650f9)

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

L02. Missing Array Length Check

Impact	Low
Likelihood	Medium

The function *batchClaimBalance()* lacks the array length equality checks, which will lead to unexpected behavior if the length of arrays is different.

Path: ./contracts/MainnetRFQ.sol: *batchClaimBalance()*

Recommendation: Implement the _assets.length == _amounts.length check.

Found in: bc4b5dd

Status: Fixed (Revised commit: f13e089)

Informational

I01. Inefficient Gas Model - Loop of Storage Interactions

In the *batchClaimBalance()* function, the variable *rebalancer* is read from storage in every loop iteration.

Accessing storage variables multiple times is not very Gas efficient.

Path: ./contracts/MainnetRFQ.sol: *batchClaimBalance()*

Recommendation: Read *rebalancer* variable to memory and use the memory variable inside the while loop.

Found in: f8881f9

Status: Fixed (Revised commit: 4d650f9)

I02. Functions that Can Be Declared External

“public” functions that are never called by the contract should be declared “external” to save Gas.

Path: ./contracts/MainnetRFQ.sol: *intialize()*

Recommendation: Use the external attribute for functions never called from the contract.

Found in: f8881f9

Status: Fixed (Revised commit: 4d650f9)

I03. Boolean Equality

Boolean constants can be used directly and do not need to be compared to true or false.

Path: ./contracts/MainnetRFQ.sol: *simpleSwap()*, *claimBalance()*, *batchClaimBalance()*

Recommendation: Remove boolean equality.

Found in: f8881f9

Status: Fixed (Revised commit: e2cfd50)

I04. Duplicate Code

The check if the caller is the rebalancer is repeated several times instead of being used in a modifier.

```
require(msg.sender == rebalancer, "RF-OCR-01");
```

Repeating require statements throughout the contract code can lead to unnecessary code duplication. This can make the codebase harder to maintain and more prone to errors.

Path: ./contracts/MainnetRFQ.sol: *claimBalance()*, *batchClaimBalance()*, *receive()*

Recommendation: Use a modifier instead of repeating require statements. It will make code more maintainable, consistent and readable, while potentially improving Gas efficiency.

Found in: f8881f9

Status: Fixed (Revised commit: 4d650f9)

I05. Solidity Style Guides 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:

- Type declarations
- State variables
- Events
- Modifiers
- Functions

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

- Constructor
- Receive function (if exists)
- Fallback function (if exists)

- External functions
- Public functions
- Internal functions
- Private functions

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 are strongly recommended. These measures aid in the comprehension of code and enhance overall code quality.

Path: ./contracts/MainnetRFQ.sol

Recommendation: Consistent adherence to the official Solidity style guide is recommended. This enhances readability and maintainability of the code, facilitating seamless interaction with the contracts. Providing comprehensive NatSpec annotations for functions and following Solidity's naming conventions further enrich the quality of the code.

Found in: bc4b5dd

Status: Fixed (Revised commit: 36bad09)

I06. Missing Events for Critical Value Updates

Events should be emitted after sensitive changes take place, to facilitate tracking and notify off-chain clients following the contract's activity.

Path:
./contracts/MainnetRFQ.sol: addTrustedContract(), removeTrustedContract().

Recommendation: Consider emitting *events* in the aforementioned functions.

Found in: 36bad09

Status: Fixed (Revised commit: 87970b8)

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/Dexalot/contracts
Commit	f8881f901e3680cdf281de7ef8e2812e4a89ec8d
Whitepaper	Link
Functional Requirements	Link
Technical Requirements	Link
Contracts	File: contracts/MainnetRFQ.sol SHA3: 334e4563a80a14c1707118924c89971eb32b9d407d94be8778597b06202d4ad8

Second review scope

Repository	https://github.com/Dexalot/contracts
Commit	4d650f9152b5c90a63a25f13c2a0176c2632526d
Whitepaper	Link
Requirements	Link
Technical Requirements	Link
Contracts	File: contracts/MainnetRFQ.sol SHA3: 36be1f2e5698e8e9b9e9c0aa7efc002d60f48d2d5eaaf83c179356b307e3c12b

Third review scope

Repository	https://github.com/Dexalot/contracts
Commit	e2cfd502dd25949661675f5f905f8506ae112477
Whitepaper	Link
Requirements	Link
Technical Requirements	Link
Contracts	File: contracts/MainnetRFQ.sol SHA3: 94c7dc33ae76ba2502a07fd48760687ff4b1aa11799aad1186c2d9b7011b0a1b

Fourth review scope

Repository	https://github.com/Dexalot/contracts
Commit	bc4b5dd230259edd0aeb521fda3053493b4701c4
Whitepaper	Link
Requirements	Link
Technical Requirements	Link
Contracts	File: contracts/MainnetRFQ.sol SHA3: 90d22fd135ecc59a890d0faee878d516cd68dfd2dfb595f963efc953c509390a

Fifth review scope

Repository	https://github.com/Dexalot/contracts
Commit	f13e0898f3e9005bcae01f39fbc1f222528e8382
Whitepaper	Link
Requirements	Link
Technical Requirements	Link
Contracts	File: contracts/MainnetRFQ.sol SHA3: f369b89f93b631d886f03cb6e4aac2ad24632ffcba6a4117c04a9bb10c55d915

Sixth review scope

Repository	https://github.com/Dexalot/contracts
Commit	36bad0968f4f6e7d4521c16949b99b5aa04da688
Whitepaper	Link
Requirements	Link
Technical Requirements	Link
Contracts	File: contracts/MainnetRFQ.sol SHA3: c6e5f92459320e9e32f0da836eb4e323b1a76ea2a611b1d1b1b66a52

Seventh review scope

Repository	https://github.com/Dexalot/contracts
Commit	87970b8cc581b59880102b327c1a61eeea0ccd88

Whitepaper	Link
Requirements	Link
Technical Requirements	Link
Contracts	File: contracts/MainnetRFQ.sol SHA3: d3b001ff08dee983eb789b35ebcb00b7fa37713119768c0639019ac5c4fc5525