

**HACKEN**

# SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

---

**Customer:** Dexalot

**Date:** September 15<sup>th</sup>, 2021

This document 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 containing confidential information can be used internally by the Customer, or it can be disclosed publicly after all vulnerabilities are fixed – upon a decision of the Customer.

## Document

<b>Name</b>	Smart Contract Code Review and Security Analysis Report for Dexalot.
<b>Approved by</b>	Andrew Matiukhin   CTO Hacken OU
<b>Type</b>	Exchange; Portfolio; Fee; OrderBooks; TradePairs
<b>Platform</b>	Ethereum / Solidity
<b>Methods</b>	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
<b>Zip archive</b>	contracts-574e192e78a45df9df3a0331cbe307c6bb023852.zip
<b>Technical Documentation</b>	YES
<b>JS tests</b>	YES
<b>Timeline</b>	26 AUGUST 2021 - 15 SEPTEMBER 2021
<b>Changelog</b>	03 SEPTEMBER 2021 - INITIAL AUDIT 15 SEPTEMBER 2021 - SECOND REVIEW



## Table of contents

Introduction	4
Scope	4
Executive Summary	5
Severity Definitions	7
Audit overview	8
Conclusion	9
Disclaimers	11

## 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 contract and its code review conducted between August 26<sup>th</sup>, 2021 - September 3<sup>rd</sup>, 2021. The second code review conducted on September 15<sup>th</sup>, 2021.

## Scope

The scope of the project is smart contracts in the repository:

**Zip archive:**

[contracts-574e192e78a45df9df3a0331cbe307c6bb023852.zip](#)

**md5 hash:**

[33109fbc487dbfee54aa07593d63c36f](#)

**Technical Documentation:** Yes

**JS tests:** Yes

**Contracts:**

- [interfaces\IPortfolio.sol](#)
- [interfaces\ITradePairs.sol](#)
- [library\Bytes32Library.sol](#)
- [library\Bytes32LinkedListLibrary.sol](#)
- [mock\MockToken.sol](#)
- [library\RBTLibrary.sol](#)
- [library\StringLibrary.sol](#)
- [Exchange.sol](#)
- [Fee.sol](#)
- [OrderBooks.sol](#)
- [Portfolio.sol](#)
- [TradePairs.sol](#)

We have scanned this smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that are considered:

Category	Check Item
Code review	<ul style="list-style-type: none"> <li>Reentrancy</li> <li>Ownership Takeover</li> <li>Timestamp Dependence</li> <li>Gas Limit and Loops</li> <li>DoS with (Unexpected) Throw</li> <li>DoS with Block Gas Limit</li> <li>Transaction-Ordering Dependence</li> <li>Style guide violation</li> <li>Costly Loop</li> <li>ERC20 API violation</li> <li>Unchecked external call</li> <li>Unchecked math</li> <li>Unsafe type inference</li> <li>Implicit visibility level</li> <li>Deployment Consistency</li> <li>Repository Consistency</li> <li>Data Consistency</li> </ul>
Functional review	<ul style="list-style-type: none"> <li>Business Logics Review</li> <li>Functionality Checks</li> <li>Access Control &amp; Authorization</li> <li>Escrow manipulation</li> <li>Token Supply manipulation</li> <li>Assets integrity</li> <li>User Balances manipulation</li> <li>Data Consistency manipulation</li> <li>Kill-Switch Mechanism</li> <li>Operation Trails &amp; Event Generation</li> </ul>

## Executive Summary

According to the assessment, the Customer's smart contracts are well-secured.





Our team performed an analysis of code functionality, manual audit, and automated checks with Mythril and Slither. All issues found during automated analysis were manually reviewed, and important vulnerabilities are presented in the Audit overview section. All found issues can be found in the Audit overview section.

As a result of the audit, security engineers found **5** low severity issues.

After the second review security engineers found all issues were **resolved**.



## Severity Definitions

Risk Level	Description
<b>Critical</b>	Critical vulnerabilities are usually straightforward to exploit and can lead to assets loss or data manipulations.
<b>High</b>	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
<b>Medium</b>	Medium-level vulnerabilities are important to fix; however, they can't lead to assets loss or data manipulations.
<b>Low</b>	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that can't have a significant impact on execution

## Audit overview

### ■ ■ ■ ■ Critical

No critical issues were found.

### ■ ■ ■ High

No high severity issues were found.

### ■ ■ Medium

No medium severity issues were found.

### ■ Low

#### 1. No events on setPortfolio function

The function setPortfolio updates a critical contract value therefore should emit an event for better tracking off-chain.

**Recommendation:** Please emit an event when changing the portfolio value.

**Fixed before the second review**

#### 2. No events on setTradePairs function

The function setTradePairs updates a critical contract value therefore should emit an event for better tracking off-chain.

**Recommendation:** Please emit an event when changing the tradePairs value.

**Fixed before the second review**

#### 3. Implicit state variable visibility

When visibility is not explicitly declared it is assumed to be internal. But it could be unclear to reviewers.

**Recommendation:** Please add an explicit visibility declaration.

**Fixed before the second review**

#### 4. Reading state variable in the loop

Calling `length()` method of the `EnumerableSetUpgradeable` for the state variable is burning gas.

**Recommendation:** Please store result of the `length()` call to the local variable and use it in the loop.

**Fixed before the second review**

#### 5. Multiple access for the state variable

Accessing the state variable in the function multiple times just burns the gas.





**Recommendation:** Please store the value of the state variable in the local variable.

**Fixed before the second review**



## Conclusion

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools.

The audit report contains all found security vulnerabilities and other issues in the reviewed code.

As a result of the audit, security engineers found **5** low severity issues.

After the second review security engineers found all issues were **resolved**.



## Disclaimers

### **Hacken Disclaimer**

The smart contracts given for audit have been analyzed in accordance with the best industry practices at the date of this report, in relation to 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 as a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. 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 can't guarantee the explicit security of the audited smart contracts.