SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Dexalot
Date: September 3rd, 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

Name	Smart Contract Code Review and Security Analysis Report for Dexalot.
Approved by	Andrew Matiukhin | CTO Hacken OU
Type	Exchange; Portfolio; Fee; OrderBooks; TradePairs
Platform	Ethereum / Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Zip archive	contracts-cbf43fa4459799ec00325495868ad155e2e84e70.zip
Technical Documentation	YES
JS tests	YES
Timeline	26 AUGUST 2021 - 03 SEPTEMBER 2021
Changelog	03 SEPTEMBER 2021 - Initial Audit

Table of contents

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 26th, 2021 - September 3rd, 2021.

Scope

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

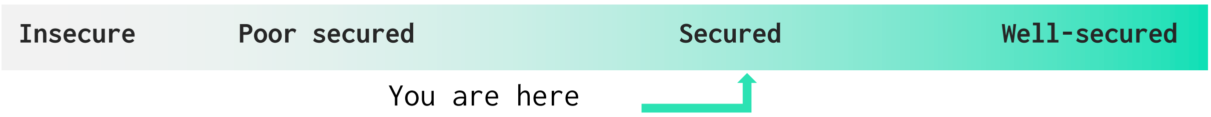
Zip archive:
contracts-cbf43fa4459799ec00325495868ad155e2e84e70.zip
md5 hash:
101a3979e67a10437a12494542cddb35
Technical Documentation: Yes
JS tests: Yes
Contracts:
interfaces\IPortfolio.sol
interfaces\ITradePairs.sol
library\Bytes32Library.sol
library\Bytes32LinkedListLibrary.sol
library\MockToken.sol
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	▪ Reentrancy ▪ Ownership Takeover ▪ Timestamp Dependence ▪ Gas Limit and Loops ▪ DoS with (Unexpected) Throw ▪ DoS with Block Gas Limit ▪ Transaction-Ordering Dependence ▪ Style guide violation ▪ Costly Loop ▪ ERC20 API violation ▪ Unchecked external call ▪ Unchecked math ▪ Unsafe type inference ▪ Implicit visibility level ▪ Deployment Consistency ▪ Repository Consistency ▪ Data Consistency
Functional review	▪ Business Logics Review ▪ Functionality Checks ▪ Access Control & Authorization ▪ Escrow manipulation ▪ Token Supply manipulation ▪ Assets integrity ▪ User Balances manipulation ▪ Data Consistency manipulation ▪ Kill-Switch Mechanism ▪ Operation Trails & Event Generation

Executive Summary

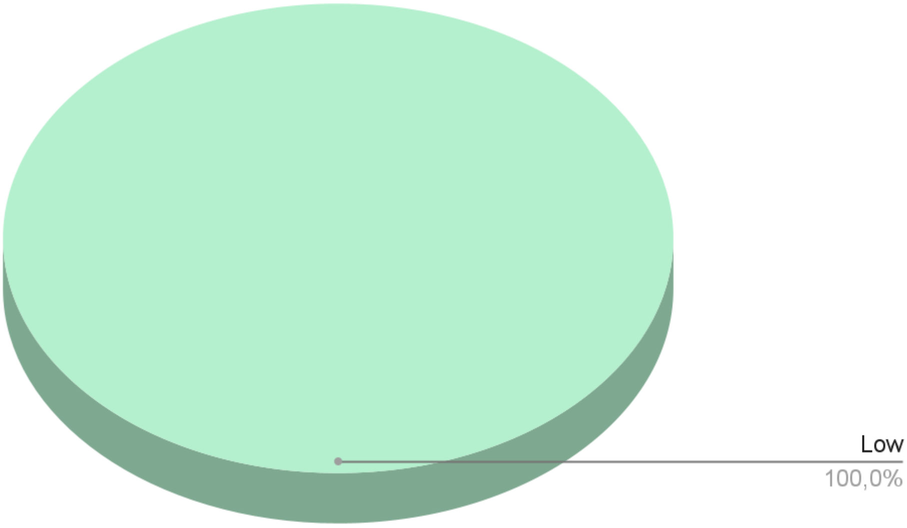
According to the assessment, the Customer's smart contracts are secured.



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.

Graph 1. The distribution of vulnerabilities after the audit.



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 can't lead to assets loss or data manipulations.
Low	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.

Lines: Exchange.sol#105-108

```
function setPortfolio(IPortfolio _portfolio) public {
   require(hasRole(DEFAULT_ADMIN_ROLE, msg.sender), "E-OACC-05");
   portfolio = _portfolio;
}
```

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.

Lines: Exchange.sol#116-119

```
function setTradePairs(ITradePairs _tradePairs) public {
   require(hasRole(DEFAULT_ADMIN_ROLE, msg.sender), "E-OACC-06");
   tradePairs = _tradePairs;
}
```

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.

Lines: Fee.sol#36-49

```
// bytes32 symbols to ERC20 token map
mapping (bytes32 => IERC20) tokenMap;

// map for numerator for share percentages
mapping (address => uint) share;
```

```
// total witdrawn by all users mapped to asset
mapping (bytes32 => uint) totalWithdrawn;

// starting total for a specific user mapped to user and asset
mapping (address => mapping (bytes32 => uint)) userTotalStart;

// total withdrawn by a specific user mapped to user and asset
mapping (address => mapping (bytes32 => uint)) userWithdrawn;
```

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.

Lines: Fee.sol#77

```
for (uint i=0; i<tokenList.length(); i++) {
```

Lines: Fee.sol#104

```
for (uint j= 0; j < tokenList.length(); j++) {
```

Lines: Fee.sol#145

```
for (uint j=0; j<tokenList.length(); j++) {
```

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.

Lines: OrderBooks.sol#69-74

```
if (orderBookMap[_orderBookID].orderBook.exists(_price)) {
    (price, parent, left, right, red) =
orderBookMap[_orderBookID].orderBook.getNode(_price);
    ( , head, ) =
orderBookMap[_orderBookID].orderList[_price].getNode('');
   size = orderBookMap[_orderBookID].orderList[_price].sizeOf();
   return (price, parent, left, right, red, head, size);
}
```

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.

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.