PALADIN
BLOCKCHAIN SECURITY

# Smart Contract Security Assessment

Final Report

## For Excalibur (Bonding)

02 February 2022

paladinsec.co

info@paladinsec.co

# Table of Contents

# Disclaimer

Paladin Blockchain Security ("Paladin") has conducted an independent audit to verify the integrity of and highlight any vulnerabilities or errors, intentional or unintentional, that may be present in the codes that were provided for the scope of this audit. This audit report does not constitute agreement, acceptance or advocation for the Project that was audited, and users relying on this audit report should not consider this as having any merit for financial advice in any shape, form or nature. The contracts audited do not account for any economic developments that may be pursued by the Project in question, and that the veracity of the findings thus presented in this report relate solely to the proficiency, competence, aptitude and discretion of our independent auditors, who make no guarantees nor assurance that the contracts are completely free of exploits, bugs, vulnerabilities or deprecation of technologies. Further, this audit report shall not be disclosed nor transmitted to any persons or parties on any objective, goal or justification without due written assent, acquiescence or approval by Paladin.

All information provided in this report does not constitute financial or investment advice, nor should it be used to signal that any persons reading this report should invest their funds without sufficient individual due diligence regardless of the findings presented in this report. Information is provided 'as is', and Paladin is under no covenant to the completeness, accuracy or solidity of the contracts audited. In no event will Paladin or its partners, employees, agents or parties related to the provision of this audit report be liable to any parties for, or lack thereof, decisions and/or actions with regards to the information provided in this audit report.

Cryptocurrencies and any technologies by extension directly or indirectly related to cryptocurrencies are highly volatile and speculative by nature. All reasonable due diligence and safeguards may yet be insufficient, and users should exercise considerable caution when participating in any shape or form in this nascent industry.

The audit report has made all reasonable attempts to provide clear and articulate recommendations to the Project team with respect to the rectification, amendment and/or revision of any highlighted issues, vulnerabilities or exploits within the contracts provided. It is the sole responsibility of the Project team to sufficiently test and perform checks, ensuring that the contracts are functioning as intended, specifically that the functions therein contained within said contracts have the desired intended effects, functionalities and outcomes of the Project team.

Paladin retains full rights over all intellectual property (including expertise and new attack or exploit vectors) discovered during the audit process. Paladin is therefore allowed and expected to re-use this knowledge in subsequent audits and to inform existing projects that may have similar vulnerabilities. Paladin may, at its discretion, claim bug bounties from third-parties while doing so.

# 1 Overview

This report has been prepared for Excalibur Exchange's Bonding contracts on the Fantom Opera network. Paladin provides a user-centred examination of the smart contracts to look for vulnerabilities, logic errors or other issues from both an internal and external perspective.

## 1.1 Summary

| | |
|---|---|
| **Project Name** | Excalibur Exchange (Bonding) |
| **URL** | https://excalibur.exchange/ |
| **Platform** | Fantom Opera |
| **Language** | Solidity |

## 1.2 Contracts Assessed

| Name | Contract | Live Code Match |
|---|---|---|
| `Bonding` | `Dependency` | MATCH |
| `BondingFactory` | `0xFEC996F9dd797A670fcb218920b6209DEf49B049` | MATCH |

## 1.3 Findings Summary

| Severity | Found | Resolved | Partially Resolved | Acknowledged (no change made) |
|---|---|---|---|---|
| High | 1 | 1 | - | - |
| Medium | 1 | 1 | - | - |
| Low | 0 | - | - | - |
| Informational | 8 | 8 | - | - |
| **Total** | **10** | **10** | **-** | **-** |

### Classification of Issues

| Severity | Description |
|---|---|
| High | Exploits, vulnerabilities or errors that will certainly or probabilistically lead towards loss of funds, control, or impairment of the contract and its functions. Issues under this classification are recommended to be fixed with utmost urgency. |
| Medium | Bugs or issues that may be subject to exploit, though their impact is somewhat limited. Issues under this classification are recommended to be fixed as soon as possible. |
| Low | Effects are minimal in isolation and do not pose a significant danger to the project or its users. Issues under this classification are recommended to be fixed nonetheless. |
| Informational | Consistency, syntax or style best practices. Generally pose a negligible level of risk, if any. |

### 1.3.1 Bonding

| ID | Severity | Summary | Status |
|---|---|---|---|
| 01 | MEDIUM | Second deposits at a later time take less than the vesting period to vest | RESOLVED |
| 02 | INFO | Contract contains unused functionality | RESOLVED |
| 03 | INFO | `factory` can be made `immutable` | RESOLVED |
| 04 | INFO | Typographical error | RESOLVED |
| 05 | INFO | Best practices should be observed | RESOLVED |
| 06 | INFO | Lack of validation | RESOLVED |

### 1.3.2 BondingFactory

| ID | Severity | Summary | Status |
|---|---|---|---|
| 07 | HIGH | Removing an element while looping over an array might revert if multiple elements are deleted | RESOLVED |
| 08 | INFO | Contract contains unused functionality | RESOLVED |
| 09 | INFO | `grailToken` and `treasury` can be made `immutable` | RESOLVED |
| 10 | INFO | Several variables can be made constant | RESOLVED |

# 2 Findings

## 2.1 Bonding

The Bonding Contract is inspired by the Olympus DAO bonds mechanism. However, it is completely developed ground up and is not forked from the Olympus DAO protocol. Users can deposit a configured token into the Bonding contract and will in turn receive vesting GRAIL tokens over time for the next few days. The idea is that users can provide LP tokens to the protocol and receive discounted but vested GRAIL tokens in return. It essentially allows users to buy GRAIL futures at a discount.

The contract specifies a maximum deposit amount which is the maximum that all users combined can deposit. There is no lower per-user maximum which means a single user could take the whole remainder.

### 2.1.1 Privileged Roles

The following functions can be called by the owner of the contract:

- `updateRatio`
- `activate`

## 2.1.2 Issues & Recommendations

| Issue #01 | Second deposits at a later time take less than the vesting period to vest |
|---|---|
| **Severity** | MEDIUM SEVERITY |
| **Location** | Line 150<br>`pending = user.totalDueRewardsAmount.sub(user.rewardDebt);` |
| **Description** | When a user deposits for a second time into the protocol, the vesting period of this second deposit is shortened due to the current vesting mechanism. We expect this to be unintentional as we assume the developer intended to make each vesting period equal to the vesting period set in the contract. |
| **Recommendation** | A redesign of the vesting mechanism will be required to fix this issue. We recommend the client refer to the Olympus DAO BondDepository contracts for inspiration. |
| **Resolution** | RESOLVED<br>The client now follows the method implemented by Olympus DAO. Through client interactions, we realized this method is indeed not ideal either as it delays the first deposit, however, it is the standard for now. |

| Issue #02 | Contract contains unused functionality |
|---|---|
| **Severity** | INFORMATIONAL |
| **Location** | Line 4<br>`import "@openzeppelin/contracts/access/Ownable.sol";` |
| **Description** | The contract contains a section of code which is not used. This can be confusing to third-party code reviewers and can make the code less accessible. The following section of code can therefore be removed. |
| **Recommendation** | Consider removing the line of code in an effort to keep the contract as simple as possible. |
| **Resolution** | RESOLVED |

| Issue #03 | `factory` can be made immutable |
| --- | --- |
| **Severity** | INFORMATIONAL |
| **Description** | Variables that are only set in the constructor but never modified can be indicated as such with the immutable keyword. This is considered best practice since it makes the code more accessible for third-party reviewers and saves gas. |
| **Recommendation** | Consider making the aforementioned variables explicitly immutable. |
| **Resolution** | RESOLVED |

| Issue #04 | Typographical error |
| --- | --- |
| **Severity** | INFORMATIONAL |
| **Location** | Line 92<br>`require(msg.sender == owner(), "isOwner: caller is not the owner");` |
| **Description** | The contract has a typographical error at the above line of code. |
| **Recommendation** | Consider fixing the typographical error. |
| **Resolution** | RESOLVED |

| Issue #05 | Best practices should be observed |
| --- | --- |
| **Severity** | INFORMATIONAL |
| **Description** | Line 115<br>`return activated && startTime <= currentBlockTimestamp && depositEndTime() > currentBlockTimestamp && totalDepositAmount < maxDepositAmount;`<br><br>Line 244<br>`if (!canHarvestBeforeEnd && currentBlockTimestamp < rewardsEndTime()) return;`<br><br>When developing smart contracts with multiple periods, each period should be as distinct as possible from each other to avoid overlap as overlapping could allow multiple actions to be executed within the same block and potentially in an unexpected order. It is therefore a best practice to clearly separate periods. Consider using <= to postpone the final harvest to right after the end time.<br><br>The exception of this best practice is the following code section.<br><br>Line 131<br>`if (_currentBlockTimestamp() < depositEndTime()) return calculateRewards(maxDepositAmount);`<br><br>Here, we believe it would be more defensive to use <= as it would return a less favorable amount for longer and therefore reduce risks. |
| **Recommendation** | Consider using<br>- `startTime < currentBlockTimestamp`<br>- `currentBlockTimestamp <= rewardsEndTime()`<br>- `_currentBlockTimestamp() <= depositEndTime()` |
| **Resolution** | RESOLVED |

| Issue #06 | Lack of validation |
| --- | --- |
| **Severity** | INFORMATIONAL |
| **Description** | The contract contains sections of code which lack proper validation. This could cause errors in case unexpected inputs are provided.<br><br>Line 225<br>`function activate() external onlyOwner {`<br><br>`activate` should only be possible before `startTime` and should only be callable once. |
| **Recommendation** | Consider using the following requirements.<br><br>`require(_currentBlockTimestamp() < startTime, ""); require(!activated, "");` |
| **Resolution** | RESOLVED |

## 2.2 BondingFactory

The BondingFactory is the main contract used by governance to create new Bonds. Each bond is defined by a set of periods, a discounted price for GRAIL and the LP token that must be supplied. The main function which the governance will periodically be calling is `createBonding` to deploy new bond instances.

The protocol has an extra safeguard such that at any point in time, all active bonds cannot increase the total GRAIL supply by more than 10%. This should reduce the risk of any detrimental issues if a bond is misconfigured.

### 2.2.1 Privileged Roles

The following functions can be called by the owner of the contract:

- `createBonding`
- `transferOwnership`
- `renounceOwnership`

## 2.2.2 Issues & Recommendations

| Issue #07 | **Removing an element while looping over an array might revert if multiple elements are deleted** |
|---|---|
| **Severity** | HIGH SEVERITY |
| **Location** | Line 147<br>`for (uint256 i = 0; i < indexesToRemove.length; i++){` |
| **Description** | The contract contains a `for` loop that removes elements from the list it is looping over. This is extremely undesirable as the list is changed and the loop indices no longer makes sense when removing an element.<br><br>Specifically, if the `indexesToRemove` contains the last element apart from an earlier element, it will revert. |
| **Recommendation** | Consider looping from the back of the array to the front to avoid this issue. This is because only the indexes at the end are changed, which would already have been handled when looping from back to front. |
| **Resolution** | RESOLVED<br><br>The client now iterates over the array back-to-front which avoids this issue. |

| Issue #08 | **Contract contains unused functionality** |
|---|---|
| **Severity** | INFORMATIONAL |
| **Description** | The contract contains a section of code which is not used. This can be confusing to third-party code reviewers and can make the code less accessible. The following section of code can therefore be removed.<br><br>Line 5<br>import "@openzeppelin/contracts/utils/ReentrancyGuard.sol"; |
| **Recommendation** | Consider removing the aforementioned line of code in an effort to keep the contract as simple as possible. |
| **Resolution** | RESOLVED |

| Issue #09 | `grailToken` and `treasury` can be made `immutable` |
|---|---|
| **Severity** | INFORMATIONAL |
| **Description** | Variables that are only set in the constructor but never modified can be indicated as such with the immutable keyword. This is considered best practice since it makes the code more accessible for third-party reviewers and saves gas. |
| **Recommendation** | Consider making the above variables explicitly `immutable`. |
| **Resolution** | RESOLVED |

| Issue #10 | Several variables can be made `constant` |
|---|---|
| **Severity** | INFORMATIONAL |
| **Description** | Variables that are never modified can be indicated as such with the constant keyword:<br>- `MAX_MINTABLE_GRAIL_PERCENT`<br>- `MIN_VESTING_PERIOD`<br>- `MIN_DEPOSIT_DURATION`<br><br>This is considered best practice since it makes the code more accessible for third-party reviewers and saves gas. |
| **Recommendation** | Consider making the variables explicitly constant. |
| **Resolution** | RESOLVED |


PALADIN
BLOCKCHAIN SECURITY