Audit Report

Braintrust Uniswap Integration

December 15, 2021

Version 1.0

Table of Contents

License

THIS WORK IS LICENSED UNDER A CREATIVE COMMONS ATTRIBUTION-NODERIVATIVES 4.0 INTERNATIONAL LICENSE.

Disclaimer

THE CONTENT OF THIS AUDIT REPORT IS PROVIDED “AS IS”, WITHOUT REPRESENTATIONS AND WARRANTIES OF ANY KIND.

THE AUTHOR AND HIS EMPLOYER DISCLAIM ANY LIABILITY FOR DAMAGE ARISING OUT OF, OR IN CONNECTION WITH, THIS AUDIT REPORT.

COPYRIGHT OF THIS REPORT REMAINS WITH THE AUTHOR.

This audit has been performed by

Oak Security

https://oaksecurity.io/
info@oaksecurity.io

Introduction

Purpose of this Report

Oak Security has been engaged by the Braintrust Technology Foundation to perform a security audit of the Braintrust Uniswap integration.

The objectives of the audit are as follows:

1. Determine the correct functioning of the contract, in accordance with the project specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine smart contract bugs, which might lead to unexpected behaviour.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebase Submitted for the Audit

The audit has been performed on the following directory in a GitHub repository:

https://github.com/Snowfork/braintrust-fee-converter/tree/main/hardhat/contracts

Commit hash: `fb30bb16b3f76514e571cd808d23881c1b86bb82`

Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line by line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under-/overflow issues
    c. Key management vulnerabilities
4. Report preparation

Functionality Overview

The submitted smart contract consists of an adapter allowing the conversion of USDC into BTRST tokens. To this end, the Uniswap v3 API is used to connect to the Uniswap router and convert from one token to another. Whilst the contract is meant to be used in combination with the USDC and BTRST smart contracts, the contract's constructor is parameterized and could be used with other pairs.

How to read this Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following: **Pending,** **Acknowledged** or **Resolved**. Informational notes do not have a status, since we consider them optional recommendations.

Note, that audits are an important step to improve the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note, that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than a security audit and vice versa.

Summary of Findings

No	Description	Severity	Status
1	Lack of deadline parameter exposes user to risk of price volatility	Minor	Resolved
2	Version pragma allows for versions with security vulnerabilities	Informational	Resolved
3	Outdated comment	Informational	Resolved
4	Contract does not implement slippage control	Informational	Acknowledged

Code Quality Criteria

Criteria	Status	Comment
Code complexity	Low	-
Code readability and clarity	High	-
Level of documentation	High	-
Test coverage	High	-

Detailed Findings

1. Lack of deadline parameter exposes user to risk of price volatility

Severity: Minor

The `deadline` parameter for the `ExactInputSingleParams` call to the Uniswap router is set to `block.timestamp`. This means that no deadline is set, implying that depending on gas provided by the frontend and network congestion, a long time might pass before the swap is executed, exposing the user to the risk of price changes. This is mitigated by carefully choosing the `amountOutMinimum`. However, for the given use case of the contract acting as a simple wrapper for off-chain code, passing on a front-end defined deadline value can help to protect users from misconfiguring this value.

Recommendation

For the given use case of the contract acting as a simple wrapper contract around the Uniswap router, we recommend adding a `deadline` parameter that users can set. The Uniswap frontend uses unix timestamp + 20 minutes as a default value.

Status: Resolved

2. Version pragma allows for versions with security vulnerabilities

Severity: Informational

The codebase allows any Solidity compiler version from 0.8 to less than 0.9. A number of important compiler bugs have recently been fixed. In particular, version 0.8.4 fixed important security issues.

Recommendation

Consider locking the version pragma to a specific compiler version, preferably 0.8.4 or above.

Status: Resolved

3. Outdated comment

Severity: Informational

The comment in line 26 and 27 does not correspond to the current implementation.

Recommendation

Consider correcting or removing the comment.

Status: Resolved

4. Contract does not implement slippage control

Severity: Informational

The contract is designed to accept an `amountOutMin` parameter that may be calculated off-chain and passed to the contract. While not a security concern, this architecture puts the responsibility of correct parameter choices on the frontend/user.

Recommendation

Ensure calculations in the front-end set the parameter to an acceptable value to reduce the risk of slippage losses.

Status: Acknowledged

The team has confirmed that they intend to handle slippage management in the front-end.