Depth Audit Report

Version 3.0.0

Serial No. 2021030200012013

Presented by Fairyproof

March 2, 2021

FAIRYPROOF

01. Introduction

This document includes the results of the audit performed by the Fairyproof team on the depth project, at the request of the depth team.

The audited code can be found in the public depth Github repository, and the versions used for this report are commits

dc6d694548d9789bdf97ca1c23eb809eced4c860,

de5199b1e35ad45930a5f2d4350cb472816d9a46,

c7934a52e512c42fec81cfd1521078a17acf9dc8,

2c41c75b53ce1e0c7a35cff79e054134e86563b1,

f09eab9a2648f2df60fe94ba273fe00fdf20fb92 and

41a3c2f75936628481c8608c40b8bf022b33d780

The goal of this audit is to review depth's solidity and vyper implementations for a decentralized exchange, study potential security vulnerabilities, its general design and architecture, and uncover bugs that could compromise the software in production.

We make observations on specific areas of the code that present concrete problems, as well as general observations that traverse the entire codebase horizontally, which could improve its quality as a whole.

— Disclaimer

Note that as of the date of publishing, the contents of this report reflect the current understanding of known security patterns and state of the art regarding smart contract security. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk.

The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. Risks or issues introduced by using data feeds from offchain sources are not extended by this review either.

Given the size of the project, the findings detailed here are not to be considered exhaustive, and further testing and audit is recommended after the issues covered are fixed.

To the fullest extent permitted by law, we disclaim all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

We do not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services.

FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

— Methodology

Depth's codebase was studied in detail in order to acquire a clear impression of how the its specifications were implemented. The codebase was then subject to deep analysis and scrutiny, resulting in a series of observations. The problems and their potential solutions are discussed in this document and, whenever possible, we identify common sources for such problems and comment on them as well.

The Fairyproof auditing process follows a routine series of steps:

1. Code review that includes the following
   i. Review of the specifications, sources, and instructions provided to Fairyproof to make sure we understand the size, scope, and functionality of the project's smart contracts.
   ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Fairyproof describe.
2. Testing and automated analysis that includes the following:
   i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run the test cases.
   ii. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve maintainability, security, and control based on the established industry and academic practices, recommendations, and research.

— Structure of the document

This report contains a list of issues and comments on all the contract files under the directory https://github.com/depthfinance/contract. Each issue is assigned a severity level based on the potential impact of the issue and recommendations to fix it, if applicable. For ease of navigation, an index by topic and another by severity are both provided at the beginning of the report.

— Documentation

For this audit, we used the following sources of truth about how the depth system should work:

https://github.com/depthfinance

These were considered the specification, and when discrepancies arose with the actual code behavior, we consulted with the depth team or reported an issue.

— Comments from Auditor

No vulnerabilities with medium severity were found in the depth's codebase. One vulnerability with critical severity, one vulnerability with high severity and one vulnerability with low severity were fixed by the team.

Two vulnerabilities with high severity were acknowledged by the team, and the team commits to transferring the owner right to a DAO or a multi-sig wallet and commits not to call the migration functions to avoid trigging the issues.

Two vulnerabilities with low severity were acknowledged by the team, and the team doesn't think they will trigger issues or risks and may make changes in future upgrades.

02. About Fairyproof

Fairyproof is a leading technology firm in the blockchain industry, providing consulting and security audits for organizations. Fairyproof has developed industry security standards for designing and deploying smart contract systems.

03. Severity level reference

Every issue in this report was assigned a severity level from the following:

Critical severity issues need to be fixed as soon as possible.

High severity issues will probably bring problems and should be fixed.

Medium severity issues could potentially bring problems and should eventually be fixed.

Low severity issues are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

04. List of issues by severity

A. Critical

- StableSwapCompound.vy

Missing Check for Function Caller's Right

B. High

- DepthToken.sol

Inappropriate Owner Right

- Breeder.sol

Logic Vulnerability and Inappropriate Owner Right

Missing Address Validiation

C. Medium

- N/A

D. Low

- DepositFilda.vy

Redundant Code

- StableSwapFilda.vy

Inappropriate Naming of Variable

- StableSwapCompound.vy

Missing Check for Zero Address

05. List of issues by contract file

- DepositFilda.vy

Redundant Code: Low

- StableSwapFilda.vy

Inappropriate Naming of Variable: Low

- StableSwapCompound.vy

Missing Check for Zero Address: Low

Missing Check for Function Caller's Right: Critical

- DepthToken.sol

Inappropriate Owner Right: High

- Breeder.sol

Logic Vulnerability and Inappropriate Owner Right: High

06. Issue descriptions and recommendations by contract file

- DepositFilda.vy

Redundant Code: Low

Source and Description:

Line 181: the function `_xp_mem` is never called or imported, and therefore it is redundent.

Line 50: the constant `PRECISION` is only used in the function `_xp_mem` which is redundent, and therefore this constant is redundent as well.

Recommendation:

Consider commenting out the function `_xp_mem` and the constant `PRECISION`.

Update: Acknowledged by the depth team. The team renamed this contract file to `DepositCompound.vy` in de5199b1e35ad45930a5f2d4350cb472816d9a46. The team doesn't think this will cause potential issues or risks and therefore prefers to keep it for now, and may make a change in a future upgrade.

- StableSwapFilda.vy

Inappropriate Naming of Variable: Low

Source and Description:

Line 640: in the function `ramp_A`, the local variable `_initial_A` actually refers to the current value of `A`. The variable is not named in a way that describes its meaning. This causes reader confusions.

Recommendation:

Consider renaming `_initial_A` to `current_A`.

Update: Acknowledged by the depth team. The team renamed this contract file to `StableSwapCompound.vy` in de5199b1e35ad45930a5f2d4350cb472816d9a46. The team doesn't think this will cause potential issues or risks and therefore prefers to keep it for now, and may make a change in a future upgrade.

- StableSwapCompound.vy

Missing Check for Zero Address: Low

Source and Description:

Line 141 in de5199b1e35ad45930a5f2d4350cb472816d9a46: the constructor doesn't check whether the two variables `_handle_lend_contract_address` and `_lend_contract_address` are zero addresses or not.

Recommendation:

Consider adding statements to check zero addresses. The recommended changes are as follows:

```
assert _handle_lend_contract_address != ZERO_ADDRESS
assert _lend_contract_address != ZERO_ADDRESS
```

Update: Fixed in *c7934a52e512c42fec81cfd1521078a17acf9dc8* by adopting the recommended changes.

Missing Check for Function Caller's Right: Critical

Source and Description:

Line 668 in de5199b1e35ad45930a5f2d4350cb472816d9a46: the function `set_handle_lend_contract_address` doesn't check whether its caller is qualified to call it or not.

Recommendation:

Consider adding a check for the caller's right before the statement `self.handle_lend_contract_address = _address` . The recommended change is as follows:

`assert msg.sender == self.owner`

Update: Fixed in *fddc146d6acadf3e6d6d4e14769eb3613d173e59* by adopting the recommended changes.

- DepthToken.sol

Inappropriate Owner Right: High

Source and Description:

Line 1288: the function `mint` can be exploited. The owner of the `WePiggyToken` contract can assign `MINTER_ROLE` to an operator enabling the operator to call the function `mint` to mint tokens at will.

Update: Acknowledged by the team. The team commits to gradually transferring the owner right to a DAO or a multi-sig wallet.

- Breeder.sol

Logic Vulnerability and Inappropriate Owner Right: High

Source and Description:

Line 1798: the implementation of the fufunction `migrate` cannot ensure all the lp data to be successfully migrated to a target contract. **The owner has the super right to migrate all locked assets. In case the owner is exploited or abuses its right, the locked assets will be in huge risks.**

Recommendation:

Consider not calling the functions that migrate the lp data and transferring the owner right to a DAO or a multi-sig wallet.

Update: Acknowledged by the team. The team makes sure not to call the functions `migrate`, `setMigrator` and `replaceMigrate` , and commits to transferring the owner right to a DAO or a multi-sig wallet, thus avoiding triggering this issue.

Missing Address Check: High

Source and Description:

Line 1976: the function `add` doesn't check whether or not `_lpToken` is a zero address. If `_lpToken` is assigned 0 a subsequent call to the function `updatePool` to update its pool will fail, thus causing all subsequent calls to the function `massUpdatePools` to fail.

Recommendation:

Consider adding an address check for `_lpToken` to make sure it is not a zero address.

Update: Fixed in *f09eab9a2648f2df60fe94ba273fe00fdf20fb92* by the team adopting the recommendation.

07. Recommendations to enhance the overall security

We list some recommendations in this section. They are not mandatory but will enhance the overall security of the system if they are adopted.

- Breeder.sol

1. The statement `totalAllocPoint = 0;` in Line 1721 is redundant and can be commented out since `totalAllocPoint` will be initialized by the system to 0.

Update: Fixed in *f09eab9a2648f2df60fe94ba273fe00fdf20fb92* by the team adopting the recommendation.

2. The `Breeder.sol` contract calls the functions defined in the `WePiggyToken` contract by importing the implementations. This can be done by just importing the interfaces instead of the implementations.

Update: Acknowledged by the team. The team will make changes in a future upgrade.

3. Consider using a uniform denominator for calculating rates, thus avoiding confusions.

Update: Acknowledged by the team. The team will make changes in a future upgrade.

4. It is better for the implementation of the function `emergencyWithdraw` in line 2113 to update the user's state before transferring tokens.

Update: Fixed in *f09eab9a2648f2df60fe94ba273fe00fdf20fb92* by the team adopting the recommendation.

5. Whenever the function `stake` defined in 1999 is called its `msg.sender` will be added to the pool array in line 2030. This may cause the size of the array extremely large. Consider defining a `mapping` variable to record all the `msg.senders` which have been added to the pool array such that when a `msg.sender` is recorded in the `mapping` addition of that `msg.sender` to the pool array will be skipped.

Update: Fixed in *f09eab9a2648f2df60fe94ba273fe00fdf20fb92* by the team adopting the recommendation.

6. In line 1857 the comment on the function `getPiggyPerBlock` does't describe the function's algorithm correctly, thus causing reader confusions.

Update: Fixed in *f09eab9a2648f2df60fe94ba273fe00fdf20fb92* by the team adopting the recommendation.