Depth Stable Coin Staking V2 Audit Report

Version 1.0.0

Serial No. 2021102200022014

Presented by Fairyproof

October 22, 2021

FAIRYPROOF

01. Introduction

This document includes the results of the audit performed by the Fairyproof team on the Depth Stable Coin Staking V2 application, at the request of the Depth team.

Audit Start Time:

October 13, 2021

Audit End Time:

October 14, 2021

Project Token's Name:

Depth Token

Audited Code's Github Repository:

https://github.com/depthfinance/contract/tree/main/BSCContracts

Audited Code's Github Commit Number When Audit Started:

a4bab7e0c49a3b95c9058be7f59325ae8767be79

Audited Source Files:

The calculated SHA-256 values for the audited files when the audit was done are as follows:

```
BDepMining.sol:
0x2165ae5038fb5b589a4bfbcee3ee1bd3663060f762f473e420eb2eb13466e9c8

BDepToken.sol:
0x59c6100fd427f1cce8dd0b529443840b78ad44996e5e6c577451809b058952da

SellLendPlatformToken.sol:
0x10ccdc1b9e4490d73845fcb88eb2ff1a623de838445b359ef3dbfbc457a3db87

threePools.vy:
0x032245da7bf923dff100a853020189387e86603ee6b5f338e30b23d5e1c3a5fc

FundingManager.sol:
0x6e51e304c4c430a796e77d3e8e748f0fcc8f4782c25c5a27681592b6b408a16c

dCowVault.sol:
0xd6c0e65fb109ed497e44cd8f175309e95a9d52df518dd598e18b8db4fac8f2de

dDepAlphaVault.sol:
```

```
0x34c80482732ddc4a69e5f41556a910123713e24f89a714a51de1ed432fd40443

dDepBxhVault.sol:
0x3d8016dcdbc602f4a31a76972209449fb634cba1861d5c3d2ceb52e320e4b124

dDepVenusVault.sol:
0x8e69f76a4c4d96aee73703bde21198d0dfda80b17e8253ba647b790d23ec7ec7
```

The source files audited include all the files with the extension "sol" as follows:

```
contracts/
├── BDepMining.sol
├── BDepToken.sol
├── Channels-3Pool
│   ├── SellLendPlatformToken.sol
│   └── threePools.vy
├── FundingManager.sol
└── vault
    ├── dCowVault.sol
    ├── dDepAlphaVault.sol
    ├── dDepBxhVault.sol
    └── dDepVenusVault.sol
```

The goal of this audit is to review Depth's solidity implementation for its token issuance and stable coin staking functions, 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.

This audit only applies to the specified code, software or any materials supplied by the Depth team for specified versions. Whenever the code, software, materials, settings, enviroment etc is changed, the comments of this audit will no longer apply.

— 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 system 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. If the audited source files are smart contract files, 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

The above files' code 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 source code.
   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 source code 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 above source files. 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 token issurance and stable coin staking should work:

https://depth.fi/

whitepaper

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 critical, high, medium or low-severity were found in the above source code.

Additional notice: 0.

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 blockchain applications.

03. Introduction to Depth

Depth is a stable coin based decentralized DeFi aggregator.

04. Major functions of audited code

The audited code mainly implements the following functions:

- Issuance of BDepToken

  Name: Depth Token

  Symbol: BDEP

  Precisions: 18

  Max supply: infinite
- Users deposit stable coins and stake the deposit certificates to earn rewards in DEP

05. Key points in audit

During the audit, we worked closely with the Depth team, and helped the team fix some bugs and refine some code. Here are the main details:

- Optimized Implementation to Reduce Gas Consumption

In the `BDepMining.sol` file, lines 311 and 317 can be optimized to reduce the gas consumption. Here was the code section:

```
if (enableClaimBlock <= block.number) {
    IERC20(Dep).safeTransfer(msg.sender, pending);
    if (user.pendingReward > 0) {
        IERC20(Dep).safeTransfer(msg.sender, user.pendingReward);
    }
```

Recommendation:

Consider combining the two `safeTransfers` to one. Here is a recommended change:

```
if (enableClaimBlock <= block.number) {
    //Remove code
    //IERC20(Dep).safeTransfer(msg.sender, pending);
    if (user.pendingReward > 0) {
        //Remove code
        //IERC20(Dep).safeTransfer(msg.sender, user.pendingReward);
        //Add code
        pending = pending.add(user.pendingReward);
        user.pendingReward = 0;
    }
    //Add code
    IERC20(Dep).safeTransfer(msg.sender, pending);
```

Update: it has been fixed in the latest audited code.

- Fixed A Bug in Conditional Check

In line 356 of the `threePools.vy` file, the `if` statement was incorrect. Here was the code section:

```
if cERC20(self.c_tokens[index]).balanceOf(self) *
cERC20(self.c_tokens[index]).exchangeRateStored() / PRECISION > PRECISION -
self.balances[index]:
```

Both line 188 and line 194 of the `dDepAlphaVault.sol` file should use `>=`. Here was the code section:

```
require(_after.sub(_before)>_amount, "sub flow!");
```

Line 245 of the `dDepAlphaVault.sol` file should use `>=`. Here was the code section:

```
require(_after.sub(_before)>_claimAmount, "sub flow!");
```

Line 371 of the `BDepMining.sol` file should be moved to the `if` directive in line 366. Here was the code section:

```
IERC20(Dep).safeTransfer(msg.sender, pending);
```

Recommendation:

Consider making changes as follows:

Changing line 365 of the `threePools.vy` file from:

```
if cERC20(self.c_tokens[index]).balanceOf(self) *
cERC20(self.c_tokens[index]).exchangeRateStored() / PRECISION > PRECISION -
self.balances[index]:
```

to:

```
if cERC20(self.c_tokens[index]).balanceOf(self) *
cERC20(self.c_tokens[index]).exchangeRateStored() / PRECISION > self.balances[index]:
```

Changing both line 188 and line 194 of the `dDepAlphaVault.sol` file from:

```
require(_after.sub(_before)>_amount, "sub flow!");
```

to:

```
require(_after.sub(_before)>=_amount, "sub flow!");
```

Changing line 245 of the `dDepAlphaVault.sol` file from:

```
require(_after.sub(_before)>_claimAmount, "sub flow!");
```

to:

```
require(_after.sub(_before)>=_claimAmount, "sub flow!");
```

Moving line 371 of the `BDepMining.sol` file to the `if` directive in line 366

Update: it has been fixed in the latest audited code.

- Optimized Implementation to Improve Efficiency

In line 452 of the `BDepMining.sol` file, the `_deleteQueueAt` function was redundant. Here was the code section:

```
function _deleteQueueAt(uint256 _pid,uint256 index) private {
       UnlockQueue[] storage queues = userUnlockQueues[_pid][msg.sender];
       for (uint256 i = index; i < queues.length - 1; i++) {
           queues[i] = queues[i + 1];
       }
       queues.pop();
   }
```

Line 468 could be replaced with `queues.pop()`

In line 115 of the `dDepVenusVault.sol` file, the `withdraw` function could be optimized. Here was the code section:

```
address _comptrollerAddress = VToken(vTokenAddress).comptroller();
if (IComptroller(address(_comptrollerAddress)).treasuryPercent() != 0) {

    RedeemLocalVars memory vars;
    vars.redeemAmount = _amount;

    uint feeAmount;
    uint remainedAmount;
    (vars.mathErr, feeAmount) = mulUInt(vars.redeemAmount,
IComptroller(address(_comptrollerAddress)).treasuryPercent());
    if (vars.mathErr != CarefulMath.MathError.NO_ERROR) {
        return;
    }

    (vars.mathErr, feeAmount) = divUInt(feeAmount, 1e18);
    if (vars.mathErr != CarefulMath.MathError.NO_ERROR) {
        return;
    }

    (vars.mathErr, remainedAmount) = subUInt(vars.redeemAmount, feeAmount);

    _amount = remainedAmount;
}
```

Recommendation:

Consider making changes as follows:

Deleting the `_deleteQueueAt` function in line 452 of the `BDepMining.sol` file. Replacing line 468 with `queues.pop()`

Changing the `withdraw` funcion in line 115 of the `dDepVenusVault.sol` file from:

```
address _comptrollerAddress = VToken(vTokenAddress).comptroller();
if (IComptroller(address(_comptrollerAddress)).treasuryPercent() != 0) {

    RedeemLocalVars memory vars;
    vars.redeemAmount = _amount;

    uint feeAmount;
    uint remainedAmount;
    (vars.mathErr, feeAmount) = mulUInt(vars.redeemAmount,
IComptroller(address(_comptrollerAddress)).treasuryPercent());
    if (vars.mathErr != CarefulMath.MathError.NO_ERROR) {
        return;
    }
    (vars.mathErr, feeAmount) = divUInt(feeAmount, 1e18);
    if (vars.mathErr != CarefulMath.MathError.NO_ERROR) {
        return;
    }

    (vars.mathErr, remainedAmount) = subUInt(vars.redeemAmount, feeAmount);

    _amount = remainedAmount;
}
```

to:

```
uint256 beforeVaule = IERC20(want).balanceOf(address(this));
require(VToken(vTokenAddress).redeemUnderlying(_amount) == 0, "!withdraw");
uint256 afterVaule = IERC20(want).balanceOf(address(this));
uint256 getValue = afterVaule.sub(beforeVaule);
require(getValue>0, "invalid amount, !withdraw");
```

Update: it has been fixed in the latest audited code.

06. Coverage of issues

The issues that the Fairyproof team covered when conducting the audit include but are not limited to the following ones:

- Re-entrancy Attack
- DDos Attack
- Integer Overflow
- Function Visibility
- Logic Vulnerability
- Uninitialized Storage Pointer
- Arithmetic Precision
- Tx.origin
- Shadow Variable
- Design Vulnerability
- Token Issurance
- Asset Security
- Access Control

07. 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.

08. Major areas that need attention

Based on the provided souce code the Fairyproof team focused on the possible issues and risks related to the following functions or areas.

- Integer Overflow/Underflow

We checked all the code sections, which had arithmetic operations and might introduce integer overflow or underflow if no safe libraries were used. All of them used safe libraries.

We didn't find issues or risks in these functions or areas at the time of writing.

- Access Control

We checked each of the functions that could modify a state, especially those functions that could only be accessed by "owner".

We didn't find issues or risks in these functions or areas at the time of writing.

- Parameter Setting

We checked whether or not some core parameters were incorrectly set.

We didn't find issues or risks in these functions or areas at the time of writing.

- Gas Consumption

We checked whether or not the gas consumption in the implementation could be reduced.

We found an issue and helped the GWC team fix it.

- Logic Issues

We checked whether or not there were issues in the logic design.

We found an issue and helped the GWC team fix it.

- Code Optimization

We checked whether or not the code could be optimized.

We found an issue and helped the GWC team fix it.

- Asset Security

We checked whether or not all the functions that transfer assets are safely hanlded.

We didn't find issues or risks in these functions or areas at the time of writing.

- Contract Migration/Upgrade

We checked whether or not the contract files introduce issues or risks associated with contract migration/upgrade.

We didn't find issues or risks in these functions or areas at the time of writing.

- Miscellaneous

We didn't find issues or risks in other functions or areas at the time of writing.

09. List of issues by severity

A. Critical

- N/A

B. High

- N/A

C. Medium

- N/A

D. Low

- N/A

10. List of issues by source file

- N/A

11. Issue descriptions

- N/A

12. 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.

- N/A