Dezy Finance DCA

Security Testing and Assessment

May 8, 2023

Prepared for Dezy Finance

Narya.ai

Table of Content

Summary

Overview

From April 4, 2023, to May 8, 2023, Dezy Finance engaged Narya.ai to test the security of its DCAStrategy contracts in the GitHub repository:https://github.com/DezyDefi/dezy-strategy-10 (commit 031c708cf0ec13b4372f8782843097a58bc89de9).

Project Scope

We reviewed and tested not only the smart contracts that implement the DCA strategy but also their interactions with external contracts on the live blockchain network.
There are other security-critical components of Dezy Finance, such as off-chain services and the web front end, which are not included in the scope of this smart contract security assessment. We recommend a further review of those components.

Summary of Findings

Severity	# of Findings
High	0
Medium	5
Low	5
Informational	0
Total	10

Throughout the testing, we identified 10 issues of medium, and low severity:

- 1 issue of amount precision loss in withdraw
- 1 issue of lack of numRounds and amount check in deposit
- 1 issue of lack of pause() and unpause()
- 1 issue of lack of numRounds check in deposit
- 1 issue of the return value of totalShares is incorrect
- 1 issue of fill() may be exposed to front-run attacks
- 1 issue of lastFillTime can be bypassed, so order can be filled twice within a period
- 1 issue of weird ERC20 tokens are not supported
- 1 issue of an order with a very large amount of deposit can be unfillable
- 1 issue of centralization

Disclaimer

This testing report should not be used as investment advice.

Narya.ai uses an automatic testing technique to test smart contracts' security properties and business logic rapidly and continuously. However, we do not provide any guarantees on eliminating all possible security issues. The technique has its limitations: for example, it may not generate a random edge case that violates an invariant during the allotted time. Its use is also limited by time and resource constraints.

Unlike time-boxed security assessment, Narya.ai advises continuing to create and update security tests throughout the project’s lifetime. In addition, Narya.ai recommends proceeding with several other independent audits and a public bug bounty program to ensure smart contract security.

Project Overview

DCAStrategy receives a certain number of ERC20 or ETH tokens from you and creates an order (tokenA, tokenB, period). Then fillers invest by filling existing live orders.

Tests

Tested Invariants and Properties

We relied on the Narya engine that used a smart fuzzing approach to test the following 3 invariants and 18 properties of the smart contracts.

Table 1 Tested Invariants

ID	Invariant/Property Description	Found Bug(s)
01	deposit with different (amount, rounds), check if withdraw as WETH with an expected result	2. Lack of numRounds and amount check in deposit 6. Lack of numRounds check in deposit
02	deposit with a random amount, fill it with random rounds, then withdraw the filled part as ETH or WETH, check withdraw result.	1. Precision loss in withdraw
03	deposit with a random amount, fill it with random rounds, then withdraw filled and unfilled parts as USDC, check withdraw result.	Passed
04	deposit with a random amount, fill it with random rounds, then withdraw as ETH or WETH, check withdraw result. withdrawal is divided into 2 steps: 1st step only withdraw filled part, 2nd step withdraw remaining parts.	Passed
05	deposit (USDC, WETH, 1 week) pair, check deposit result	Passed
06	deposit (USDT, WBTC, 1 month) pair, check deposit result	Passed
07	deposit (WETH, USDC, 1 week) pair and withdraw, check deposit and withdraw result	Passed
08	deposit (USDT, WETH, 1 month) pair and withdraw as ETH, check deposit and withdraw result	Passed
09	deposit (USDC, WETH, 1 week) pair, fill all rounds and withdraw, check fill and withdraw result	Passed

10	2 users deposit (USDC, WETH, 1 week) pair, 1 filler fill the pair, then 2 users withdraw. check fill and withdraw results	Passed
11	deposit (USDC, WETH, 1 week) pair with 5 rounds, 1 filler only fills the pair 3 rounds, then the user withdraws. check fill and withdraw results	Passed
12	deposit (USDC, WETH, 1 week) pair, then modify deposit pair and only fill part of rounds, then withdraw. check fill and withdraw results	Passed
13	2 users deposit (USDC, WETH, 1 week) pair. Check fill and withdraw results	Passed
14	1 user deposits both (USDC, WETH, 1 month) and (WETH, USDC, 1 month) pairs. check fill and withdraw results.	Passed
15	check modify results after withdrawing all filled and unfilled tokens.	Passed
16	Check withdraw the unfilled and filled amount separately.	Passed
17	Fill with flashloan, check fill and withdraw results.	Passed
18	Withdraw all and then deposit again, check withdraw and 2nd deposit results.	Passed
19	Invariant to check if the user's balance decreases.	Passed
20	Check if the fee collector receives the expected fee.	Passed
21	Deposit (USDC, WETH, 1 week) pair. invariant will check user asset after withdraw, make sure that user loss won’t be larger than 2%	Passed

Key Findings and Recommendations

1. Precision loss in withdraw

Severity: **Medium**

Description

In the else branch, division before multiplication leads to amount precision loss.

Code 1 contracts/DCAStrategy.sol#L704

```
    function _withdraw(
        address tokenA,
        address tokenB,
        uint32 period,
        bool onlyFilled,
        bool unwrapEth,
        address feeToken
    ) internal {
        ...

        if (!onlyFilled) {
           ...
        } else {
            uint64 unfilledRounds =
userDataMap[msg.sender][tokenA][tokenB][
                period
            ].rounds -
                (currentRound -

userDataMap[msg.sender][tokenA][tokenB][period].startRound);
            userDataMap[msg.sender][tokenA][tokenB][period] = UserData({
                startRound: currentRound,
                // Unfilled rounds
                rounds: unfilledRounds,
                amount: (userDataMap[msg.sender][tokenA][tokenB][period]
                    .amount /
                    userDataMap[msg.sender][tokenA][tokenB][period].rounds)
*
                    unfilledRounds
            });
```

```
            emit Withdraw(
                msg.sender,
                tokenA,
                tokenB,
                period,
                0,
                filledAmount,
                currentRound
            );
        }
        ...
    }
```

Impact

It may have an impact on user asset calculation, which leads to unnecessary user asset loss.

Failed Test

```
    function testDepositUsdcWethFillAndWithdrawFilled(
        uint256 _amount,
        uint64 _fillRounds,
        bool _unwrapEth
    ) public {
        if (_amount > 100000000000) {
            _amount = 100000000000;
        }
        uint64 _numRounds = 10;
        uint32 period = 2592000;
        address tokenA = address(USDC);
        address tokenB = address(WETH);

        vm.assume(_amount >= _numRounds);

        vm.startPrank(user1);
        _deposit(user1, tokenA, tokenB, period, _amount, _numRounds);
        vm.stopPrank();

        vm.startPrank(filler);
        IERC20(tokenB).approve(address(dca), MAX_UINT256);
        for (uint256 i = 0; i < _fillRounds % _numRounds; ) {
            if (!_fill(tokenA, tokenB, period)) {
                break;
            }
```

```
            unchecked {
                ++i;
            }
        }
        vm.stopPrank();

        vm.startPrank(user1);
        uint64 currentRoundBefore = dca.currentRoundMap(tokenA, tokenB,
period);
        (uint64 dataStartRound, uint64 dataRounds, uint256 dataAmount) = dca
            .userDataMap(user1, tokenA, tokenB, period);
        uint64 roundsExpected = dataRounds -
            (currentRoundBefore - dataStartRound);
        uint256 amountExpected = (dataAmount * roundsExpected) / dataRounds;
        console.log(
            "[proptest Expected] rounds: %s amount: %s",
            roundsExpected,
            amountExpected
        );

        // withdraw filled part
        _withdraw(user1, tokenA, tokenB, period, true, _unwrapEth);
        (dataStartRound, dataRounds, dataAmount) = dca.userDataMap(
            user1,
            tokenA,
            tokenB,
            period
        );
        require(
            dataRounds == roundsExpected,
            "userDataMap rounds updated not as expected"
        );
        console.log(
            "[prop test] result %s expected %s",
            dataAmount,
            amountExpected
        );
        require(
            dataAmount == amountExpected,
            "userDataMap amount updated not as expected"
        );
        vm.stopPrank();
    }
```

Recommendation

When calculating the amount in UserDataMap, do multiplication before Division.

Remediation

This issue has been fixed by Dezy Finance at commit 5ec8d4c24ffce8285f579212be048fedce7b5208.

2. Lack of numRounds and amount check in deposit

Severity: **Meidum**

Description

if numRounds > amount, amountPerRound will be zero. getAmounts will return (0, 0).

Code 2 contracts/DCAStrategy.sol#L530

```
    function _deposit(
        address tokenA,
        address tokenB,
        uint32 period,
        uint amount,
        uint64 numRounds
    ) internal {
        ...
    }
```

Impact

User will never get back his/her deposit assets any more

Failed Test

```
    function testDepositWithRoundsAndUsdcWethMonthlyAndWithdrawWrapped(
        uint256 _amount,
        uint64 _numRounds
    ) public {
        if (_amount > 100000000000) {
            _amount = 100000000000;
        }

        vm.startPrank(user1);
        if (
            _deposit(
                user1,
                address(USDC),
                address(WETH),
                2592000,
                _amount,
                _numRounds
            )
        ) {
```

```
            _withdraw(
                user1,
                address(USDC),
                address(WETH),
                2592000,
                false,
                false
            );
        }

        vm.stopPrank();
    }
```

Recommendation

Add a check in deposit function to make sure this condition always satisfies: numRounds <= amount.

Remediation

This issue has been fixed by Dezy Finance at commit 5ec8d4c24ffce8285f579212be048fedce7b5208.

3. Lack of pause() and unpause()

Severity: **Medium**

Description

The DCAStrategy contract refers to PausableUpgradeable to implement the pause feature. However, since the _pause() and _unpause() functions are internal, it is necessary to encapsulate functions similar to function pause() onlyOwner{} during development. It was not found in the DCAStrategy contract, so it is recommended to add it.

Code 3 contracts/DCAStrategy.sol#L30

```
contract DCAStrategy is
    ReentrancyGuardUpgradeable,
    UUPSUpgradeable,
    OwnableUpgradeable,
    PausableUpgradeable
{
   ...
}
```

Impact

Contract owner can not pause and unpause the contract.

Recommendation

Add functions with onlyOwner modifier to wrap _pause() and unpause() functions.

Remediation

This issue has been fixed by Dezy Finance at commit 5ec8d4c24ffce8285f579212be048fedce7b5208.

4. Check bypass on lastFillTime

Severity: **Medium**

Description

When the user calls fill() to fulfill an order, the lastFillTime is used to limit the frequency, e.g. for an order with period == weekly, it can be filled once a week. And when filling an order, the caller gets a discount.

When the fill() function is called, the order of the tokenA and tokenB arguments does not affect the actual token transfer. The problem here is that since the call to fill(tokenA,tokenB) only updates lastFillTime[tokenA][tokenB] == block.timestamp, when the user calls fill(tokenB,tokenA) later, the order can still be fulfilled, so that the order is fulfilled twice in one period instead of once.

Code 4 contracts/DCAStrategy.sol#L254

```
    function fill(
        address tokenA,
        address tokenB,
        uint32 period,
        bytes memory params
    ) public nonReentrant whenNotPaused returns (uint) {
        ...
        // Update fill time
        lastFillTime[tokenA][tokenB][period] = block.timestamp;

        // Sync amount to buy for next round
        amountToFillMap[tokenA][tokenB][period] -= amountToDeductMap[tokenA][
            tokenB
        ][period][currentRound];
        amountToFillMap[tokenB][tokenA][period] -= amountToDeductMap[tokenB][
            tokenA
        ][period][currentRound];
        ...
    }
```

Impact

As a result, the order can be filled twice within one period (correctly once), and the user will get more discounts.

Recommendation

Add missing update operation in fill function:

```
        // Update fill time
        lastFillTime[tokenA][tokenB][period] = block.timestamp;
+       lastFillTime[tokenB][tokenA][period] = block.timestamp;
```

Remediation

This issue has been fixed by Dezy Finance at commit 8a7844937acaf9de171379d69cfc79f8276b573b.

5. Weird ERC20 tokens are not supported

Severity: **Medium**

Description

The contract is designed to support arbitrary tokens, but does not support some of the weird ERC20 tokens. https://github.com/d-xo/weird-erc20/

Case 1: Fee on Transfer Token
The fee-on-transfer tokens are charged at the time of transfer, i.e. Alice sends 100 tokens to Bob and Bob actually receives 99 tokens.
In the contract, if the user deposits 100 tokens, the contract actually receives 99 tokens, but userDataMap.amount = 100, which will cause the user to fail when withdrawing tokens due to insufficient balance. Worse, the user can deplete the fee-on-transfer tokens in the contract by continuously depositing and withdrawing them.

Case 2: Rebase Token
The balance of rebase tokens is adjusted with the rebase event, i.e. alice has 100 tokens and after the rebase event, alice's balance may be adjusted to 101 or 99.
Consider Alice/Bob depositing 100 tokens each, and the contract has a balance of 200, then a rebase event occurs and the contract's balance is adjusted to 150, Alice withdraws 100 tokens, and Bob fails to withdraw due to insufficient balance.

Case 3: No Revert on Failure Token
Some tokens will return false instead of revert when the transfer fails (such as insufficient balance). Since the contract does not check the return value when calling transfer/transferfrom, the contract will regard the failed token transfer as successful.
Consider alice deposits 100 tokens into the contract, fails due to insufficient balance, but the contract records userDataMap[alice].amount = 100. After Bob successfully deposits 100 tokens into the contract, Alice immediately calls withdraw to withdraw Bob's 100 tokens.

Impact

It will cause users who use these tokens to lose assets.

Recommendation

- For Fee on Transfer Tokens, we can use balanceOf to get the actual amount of tokens received by the contract.
- For No Revert on Failure Tokens, we can use OZ's SafeERC20 library to transfer tokens.
- For Rebase Tokens, we should use the blacklist to prohibit the use of these tokens.

The best practice is to use a whitelist to limit the tokens supported by the contract.

Remediation

This issue has been confirmed by Dezy Finance. Dezy Finance will not support weird ERC20 tokens at its frontend for the safety of its end users.

6. Lack of numRounds check in deposit

Severity: **Low**

Description

0 numRounds leads to division zero exception

Code 5 contracts/DCAStrategy.sol#L530

```
    function _deposit(
        address tokenA,
        address tokenB,
        uint32 period,
        uint amount,
        uint64 numRounds
    ) internal {
        ...
    }
```

Impact

Deposit will fail and revert at this condition, which leads to gas fee loss.

Failed Test

```
function testDepositWithRoundsAndUsdcWethMonthlyAndWithdrawWrapped(
        uint256 _amount,
        uint64 _numRounds
    ) public {
        if (_amount > 100000000000) {
            _amount = 100000000000;
        }

        vm.startPrank(user1);
        if (
            _deposit(
                user1,
                address(USDC),
                address(WETH),
                2592000,
                _amount,
                _numRounds
            )
```

```
        ) {
            _withdraw(
                user1,
                address(USDC),
                address(WETH),
                2592000,
                false,
                false
            );
        }

        vm.stopPrank();
    }
```

Recommendation

Add a check for numRounds in the deposit function, make sure it is always larger than zero.

Remediation

This issue has been confirmed and mitigated by Dezy Finance at the frontend. It does not impact the safety of user funds.

7. Incorrect return value of totalShares

Severity: **Low**

Description

totalShares() is used to return the number of shares in the current round, but because the amountToDeductMap of the previous round is added incorrectly (the correct way is to subtract the amountToDeductMap of the current round), it causes totalShares() to return an incorrect value.

Consider the following scenario

Round 1: deposit 1000 numRounds 1
amountToFillMap = 1000, amountToDeductMap[2] = 1000
totalShares = 1000

Round 2:
amountToFillMap = 1000, amountToDeductMap[2] = 1000
totalShares = 1000

Round 3:
amountToFillMap = 1000-1000 = 0, amountToDeductMap[2] = 1000
totalShares = 0+1000 = 1000

In fact, in round 2 & 3, totalShares should be 0. There are two reasons for the incorrect return result:
1. the amountToDeductMap of the previous round is incorrectly added
2. the amountToDeductMap of the previous round is not cleared

Code 7 contracts/DCAStrategy.sol#L396

```
    function totalShares(
        address tokenA,
        address tokenB,
        uint32 period
    ) public view returns (uint) {
        return (amountToFillMap[tokenA][tokenB][period] +
            amountToDeductMap[tokenA][tokenB][period][
                currentRoundMap[tokenA][tokenB][period] - 1
            ]);
    }
```

Impact

totalShares() is an important function, especially when integrated with other projects, it is generally used to calculate the percentage of the user's assets, the incorrect return value of totalShares() may cause serious calculation errors when integrated with other projects

Recommendation

1. Clear the amountToDeductMap of the current round in the fill function

```
        amountToFillMap[tokenA][tokenB][period] -= amountToDeductMap[tokenA][
            tokenB
        ][period][currentRound];
        amountToFillMap[tokenB][tokenA][period] -= amountToDeductMap[tokenB][
            tokenA
        ][period][currentRound];
+       amountToDeductMap[tokenA][tokenB][period][currentRound] = 0;
+       amountToDeductMap[tokenB][tokenA][period][currentRound] = 0;
```

2. Modify the totalShares() function

```
    function totalShares(
        address tokenA,
        address tokenB,
        uint32 period
    ) public view returns (uint) {
+       return (amountToFillMap[tokenA][tokenB][period] -
+           amountToDeductMap[tokenA][tokenB][period][
+               currentRoundMap[tokenA][tokenB][period]
+           ]);
-       return (amountToFillMap[tokenA][tokenB][period] +
-           amountToDeductMap[tokenA][tokenB][period][
-               currentRoundMap[tokenA][tokenB][period] - 1
-           ]);
    }
```

Remediation

This issue has been fixed by Dezy Finance at commit 8a7844937acaf9de171379d69cfc79f8276b573b.

8. Possible front-run attacks on fill

Severity: **Low**

Description

When the fill function is called to fulfill the order, if the token distribution in the order is changed due to the execution of deposit/withdraw, the fulfiller's token transfer may be different than expected.

Case 1: fill() can be front run by deposit()/withdraw()
Consider ETH on downtrend.
ETH:USDC= 1:2000, amountToFillMap[USDC][ETH] = 3000e6, amountToFillMap[ETH][USDC] = 1e18, fee = 1%.
alice calls fill(USDC,ETH) to fill 1000 USDC with 0.495 ETH
bob wants to sell his ETH, and bob observes alice's tx, bob uses MEV to deposit 5 ETH
Alice tx is executed, fill 4.5 ETH with 8910 USDC.

Also, due to another issue - "fill() can be called twice in one period" - it leads to another case as follows

Case 2: fill() can be front run by fill()
Consider ETH on downtrend.
ETH:USDC= 1:2000, amountToFillMap[USDC][ETH] = 3000e6, amountToFillMap[ETH][USDC] = 1e18, fee = 1%.
In round 1, alice calls fill(USDC,ETH) to fill 1000 USDC with 0.495 ETH
bob wants to sell his ETH, and bob observes alice's tx, bob uses MEV to call fill(ETH,USDC) first, and deposit 5 ETH in round 2
alice'tx is executed in round 2, fill 4.5 ETH with 8910 USDC.

Code 8 **contracts/DCAStrategy.sol#L254**

```
    function fill(
        address tokenA,
        address tokenB,
        uint32 period,
        bytes memory params
    ) public nonReentrant whenNotPaused returns (uint) {
        ...
    }
```

Impact

The fulfiller's token transfer may be different than expected.

Recommendation

Consider adding the exceptedToken and maxFillAmount parameters to the fill() function to allow the fulfiller to specify the token and the amount to be fulfilled.

Remediation

This issue has been confirmed by Dezy Finance. Dezy Finance has fixed Case 2 at commit 8a7844937acaf9de171379d69cfc79f8276b573b and will resolve Case 1 in a future contract upgrade.

9. Unfillable deposit on order with a very large amount

Severity: **Low**

Description

Supposing an attacker deposits a huge amount of tokenA to create a DCA pair (tokenA, tokenB), this swap cannot be supported with the given exchange rate from the oracle, so no users want to fill this order because of no profit. This will make the demand of other users who have deposited in the same pair unfillable.

Code 9 contracts/DCAStrategy.sol#L274

```
    function fill(
        address tokenA,
        address tokenB,
        uint32 period,
        bytes memory params
    ) public nonReentrant whenNotPaused returns (uint) {
        require(
            block.timestamp >= lastFillTime[tokenA][tokenB][period] + period,
            "ERR: PERIOD_NOT_READY"
        );
        uint returnAmount;
        uint64 currentRound = currentRoundMap[tokenA][tokenB][period];
        // Gets the token and amount required for filling
        (
            address fillToken,
            uint nettFillAmount,
            uint rate,
            uint amountA,
            uint amountB,
            uint tokenADecimals
        ) = getFillPair(tokenA, tokenB, period, currentRound);

        uint fillingFeeAmount = (nettFillAmount * fillingFee) / DIVISOR;
        ...
    }
```

Impact

The affected orders will always be unfilled before the attacker withdraws. The DCA process cannot continue. The users involved in the related orders can only choose to withdraw their funds.

Recommendation

Consider supporting partial fills of orders.

Remediation

This issue has been confirmed by Dezy Finance. Dezy Finance will support partial filling in the future. This issue does not impact the safety of user funds since users can still withdraw in any case.

10. Centralization

Severity: **Low**

Description

The owner can set the fee to 100%, so that the user will lose all assets when calling withdraw().

Code 11 contracts/DCAStrategy.sol#L723

```
        // Send to user filledAmount
        if (filledAmount > 0) {
            uint fee = (filledAmount *
                feesInstance.calcFee(STRATEGY_INDEX, msg.sender, feeToken)) /
                DIVISOR;
            IERC20(tokenB).transfer(
                feesInstance.feeCollector(STRATEGY_INDEX),
                fee
            );
            _transfer(
                msg.sender,
                tokenB == wethAddress && unwrapEth ? address(0) : tokenB,
                filledAmount - fee
            );
            tokenShares[tokenB] -= filledAmount;
        }
```

Impact

It may cause users to lose assets when the owner account of the DCA contract is compromised.

Recommendation

- For case 1, while it would be a good practice to remove the whenNotPaused modifier from the withdraw function, it might make it impossible to pause the hack in case of emergency, so it can be unfixed, but users should be made aware of the risks
- For case 2, we can set an upper limit on the fee, e.g. 1%

Remediation

This issue has been fixed by Dezy Finance at commit 8a7844937acaf9de171379d69cfc79f8276b573b.

Fix Log

Table 2 Fix Log

ID	Title	Severity	Status
01	Precision loss in withdraw	**Medium**	**Fixed at commit 5ec8d4c24ffce8285f579212be048fedce7b5208**
02	Lack of numRounds and amount check in deposit	**Medium**	**Fixed at commit 5ec8d4c24ffce8285f579212be048fedce7b5208**
03	Lack of pause and unpause	**Medium**	**Fixed at commit 5ec8d4c24ffce8285f579212be048fedce7b5208**
04	Check bypass on lastFillTime	**Medium**	**Fixed at commit 8a7844937acaf9de171379d69cfc79f8276b573b**
05	Weird ERC20 tokens are not supported	**Medium**	**Confirmed and Mitigated***
06	Lack of numRounds check in deposit	**Low**	**Confirmed and Mitigated***
07	Incorrect return value of totalShares	**Low**	**Fixed at commit 8a7844937acaf9de171379d69cfc79f8276b573b**
08	Possible front-run attacks on fill	**Low**	**Partially fixed at commit 8a7844937acaf9de171379d69cfc79f8276b573b***
09	Unfillable deposit on order with a very large amount	**Low**	**Confirmed***
10	Centralization	**Low**	**Fixed at commit 8a7844937acaf9de171379d69cfc79f8276b573b**

* The issues have been confirmed by Dezy Finance but won't be fully fixed at the time of this report. We confirm that they do not impact the fund safety. Check the former description of each issue for details.

Appendix

Severity Categories

Severity	Description
Gas	Gas optimization.
Informational	The issue does not pose a security risk but is relevant to best security practices.
Low	The issue does not put assets at risk such as functions being inconsistent with specifications or issues with comments.
Medium	The issue puts assets at risk not directly but with a hypothetical attack path, stated assumptions, or external requirements. Or the issue impacts the functionalities or availability of the protocol.
High	The issue directly results in assets being stolen, lost, or compromised.