4th05

D.A.A.O

Security Review Report

31 January 2025

Security Review Report

4th05

January 31, 2025

Table of Contents

Protocol Summary

DAAO is a decentralized autonomous agentic organization protocol that enables automated fundraising and liquidity management through Uniswap V3. The protocol allows projects to raise funds through

configurable fundraising rounds (both whitelisted and public), automatically creates and locks Uniswap V3 positions with the raised funds, and implements a sophisticated treasury management system with built-in tax mechanisms and fee collection from liquidity positions. The smart contract architecture ensures secure fund management while providing flexibility for DAO governance and automated market making.

Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where I try to find as many vulnerabilities as possible. I can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs and on-chain monitoring are strongly recommended.

Risk Classification

		Impact		
		High	Medium	Low
Likelihood	High	H	H/M	M
	Medium	H/M	M	M/L
	Low	M	M/L	L

Overview

Contest platform	Cantina
LOC	517
Language	Solidity
Commit	85b759da2eda23a71032c3dac023a73e63afa167
Previous audits	LightChaser

Scope

- CLPoolRouter.sol
- Daao.sol
- DaaoToken.sol
- interface.sol

Issues found

Severity	Number of issues found
High	0
Medium	1
Low	3
Info	3

Findings

[M1] All the amounts sent through the `receive()` after deadline are lost

Summary All the amounts sent through the `receive()` after deadline are lost.

Finding Description All the contributions sent to the contract after the `fundraisingDeadline` do not contribute to `totalRaised` and are not even sent back to users.

Impact Explanation Loss of funds for the user

Likelihood Explanation Medium considering that `receive()` has been designed as a method through which contribute to the fundraising for the user and that (because of its simplicity), user may want to use it also if they are close to the deadline without running risks to loose their funds.

Recommendation

```
-    receive() external payable {
+    receive() external payable nonReentrant {
        if (!goalReached && block.timestamp < fundraisingDeadline) {
            contribute();
        }
+       else {
```

```
+         payable(msg.sender).transfer(msg.value);
+       }
    }
```

[L1] `Daoo::contribute` reverts when it could take `effectiveamount` instead

Summary

`Daoo::contribute` reverts even if it could have taken an `effectiveamount` increasing the `totalRaised`.

Finding Description

Within `Daoo::contribute` there are `require` statements related to the total contribution amount of the `msg.sender`.

```
function contribute() public payable nonReentrant {
        require(!goalReached, "Goal already reached");
        require(block.timestamp < fundraisingDeadline, "Deadline hit");
        require(msg.value > 0, "Contribution must be greater than 0");

        // Must be whitelisted
        WhitelistTier userTier = userTiers[msg.sender];
        require(userTier != WhitelistTier.None, "Not whitelisted");
        // Contribution must boolow teir limit
        uint256 userLimit = tierLimits[userTier];
    @>    require(
            contributions[msg.sender] + msg.value <= userLimit,
            "Exceeding tier limit"
        );

        if (maxWhitelistAmount > 0) {
    @>    require(
                contributions[msg.sender] + msg.value <=
                    maxWhitelistAmount,
                "Exceeding maxWhitelistAmount"
            );
        } else if (maxPublicContributionAmount > 0) {
    @>    require(
                  contributions[msg.sender] + msg.value <=
                    maxPublicContributionAmount,
                "Exceeding maxPublicContributionAmount"
            );
        }
```

All these `require` statements take into account the `msg.value` data of the transaction. However after these statements it is given the possibility to the user to contribute with just a fraction of the `msg.value` increasing this way `totalRaised` by less then `msg.value` amount. It could be that

all these `require` statements would have been satisfied with the `effectiveamount` instead of considering the whole `msg.value`. Placeing all these `require` statements before knowing the final value of `effectiveamount` prevent the contract to get additional funds satisfying all the require.

```
        uint256 effectiveContribution = msg.value;
        if (totalRaised + msg.value > fundraisingGoal) {
            effectiveContribution = fundraisingGoal - totalRaised;
            payable(msg.sender).transfer(msg.value -
                effectiveContribution);
        }

        if (contributions[msg.sender] == 0) {
            contributors.push(msg.sender);
        }

        contributions[msg.sender] += effectiveContribution;
        totalRaised += effectiveContribution;

        if (totalRaised == fundraisingGoal) {
            goalReached = true;
        }

        emit Contribution(msg.sender, effectiveContribution);
    }
```

Impact Explanation

A transaction that could be finalized increasing the `totalRaised` will be reverted with a loss of funds for the contract that is equal to the `effectiveamount`.

Likelihood Explanation

Medium as users may do not know which are their current contribution limits, considering they could potentially be changed at any time during fundraising.

Recommendation

```
function contribute() public payable nonReentrant {
        require(!goalReached, "Goal already reached");
        require(block.timestamp < fundraisingDeadline, "Deadline hit");
        require(msg.value > 0, "Contribution must be greater than 0");

        // Must be whitelisted
        WhitelistTier userTier = userTiers[msg.sender];
        require(userTier != WhitelistTier.None, "Not whitelisted");
        // Contribution must boolow teir limit
        uint256 userLimit = tierLimits[userTier];
-        require(
-            contributions[msg.sender] + msg.value <= userLimit,
-            "Exceeding tier limit"
-        );
```

```
15
16 -        if (maxWhitelistAmount > 0) {
17 -        require(
18 -                contributions[msg.sender] + msg.value <=
   maxWhitelistAmount,
19 -                "Exceeding maxWhitelistAmount"
20             );
21 -        } else if (maxPublicContributionAmount > 0) {
22 -        require(
23 -                  contributions[msg.sender] + msg.value <=
24 -                    maxPublicContributionAmount,
25 -                "Exceeding maxPublicContributionAmount"
26 -            );
27 -        }
28
29          uint256 effectiveContribution = msg.value;
30          if (totalRaised + msg.value > fundraisingGoal) {
31              effectiveContribution = fundraisingGoal - totalRaised;
32 +        require(
33 +            contributions[msg.sender] + effectiveContribution <=
   userLimit,
34 +            "Exceeding tier limit"
35 +        );
36
37 +        if (maxWhitelistAmount > 0) {
38 +        require(
39 +                contributions[msg.sender] + effectiveContribution <=
   maxWhitelistAmount,
40 +                "Exceeding maxWhitelistAmount"
41             );
42 +        } else if (maxPublicContributionAmount > 0) {
43 +        require(
44 +                  contributions[msg.sender] + effectiveContribution <=
45 +                    maxPublicContributionAmount,
46 +                "Exceeding maxPublicContributionAmount"
47 +            );
48 +        }
49              payable(msg.sender).transfer(msg.value -
                   effectiveContribution);
50          }
51 +       else {
52 +        require(
53 +            contributions[msg.sender] + msg.value <= userLimit,
54 +            "Exceeding tier limit"
55 +        );
56
57 +        if (maxWhitelistAmount > 0) {
58 +        require(
59 +                contributions[msg.sender] + msg.value <=
   maxWhitelistAmount,
60 +                "Exceeding maxWhitelistAmount"
```

```
61 +             );
62 +         } else if (maxPublicContributionAmount > 0) {
63 +         require(
64 +                   contributions[msg.sender] + msg.value <=
65 +                     maxPublicContributionAmount,
66 +                 "Exceeding maxPublicContributionAmount"
67 +             );
68 +         }
69 +        }
70
71          if (contributions[msg.sender] == 0) {
72              contributors.push(msg.sender);
73          }
74
75          contributions[msg.sender] += effectiveContribution;
76          totalRaised += effectiveContribution;
77
78          if (totalRaised == fundraisingGoal) {
79              goalReached = true;
80          }
81
82          emit Contribution(msg.sender, effectiveContribution);
83      }
```

[L2] Missing check in the `Daoo::extendFundraisingDeadline` would allow to brake a `require` constructor statement

Summary

In the `Daoo::extendFundraisingDeadline` the `owner`/`protocolAdmin` can set a new deadline > `fundExpiry` which would brake the protocol core functionality.

```
1  constructor(
2          uint256 _fundraisingGoal,
3          string memory _name,
4          string memory _symbol,
5          uint256 _fundraisingDeadline,
6          uint256 _fundExpiry,
7          address _daoManager,
8          address _liquidityLockerFactory,
9          uint256 _maxWhitelistAmount,
10         address _protocolAdmin,
11         uint256 _maxPublicContributionAmount
12     ) Ownable(_daoManager) {
13         require(
14             _fundraisingGoal > 0,
15             "Fundraising goal must be greater than 0"
16         );
17         require(
```

```
18              _fundraisingDeadline > block.timestamp,
19              "_fundraisingDeadline > block.timestamp"
20          );
21  @>        require(
22              _fundExpiry > fundraisingDeadline,
23              "_fundExpiry > fundraisingDeadline"
24          );
25          name = _name;
26          symbol = _symbol;
27          fundraisingGoal = _fundraisingGoal;
28          fundraisingDeadline = _fundraisingDeadline;
29          fundExpiry = _fundExpiry;
30          liquidityLockerFactory = ILockerFactory(_liquidityLockerFactory
                );
31          maxWhitelistAmount = _maxWhitelistAmount;
32          protocolAdmin = _protocolAdmin;
33          maxPublicContributionAmount = _maxPublicContributionAmount;
34
35          // Teir allocation
36          tierLimits[WhitelistTier.Platinum] = PLATINUM_DEFAULT_LIMIT;
37          tierLimits[WhitelistTier.Gold] = GOLD_DEFAULT_LIMIT;
38          tierLimits[WhitelistTier.Silver] = SILVER_DEFAULT_LIMIT;
39      }
```

Finding Description

In `Daoo::extendFundraisingDeadline` there is not `require` for `newFundraisingDeadline <fundExpiry`.

Impact Explanation

Would brake core functionality related to the deployment done in `Daoo::finalizeFundraising` .

```
1  address lockerAddress = liquidityLockerFactory.deploy(
2              address(POSITION_MANAGER),
3              owner(),
4  @>          uint64(fundExpiry),
5              tokenId,
6              lpFeesCut,
7              address(this)
8          );
```

Likelihood Explanation

Depends on the time window between `fundExpiry` value and the `fundraisingDeadline`. However, `owner && protocolAdmin` roles are trusted.

Recommendation

```
1  function extendFundraisingDeadline(
```

```
        uint256 newFundraisingDeadline
    ) external {
        require(
            msg.sender == owner() || msg.sender == protocolAdmin,
            "Must be owner or protocolAdmin"
        );
        require(!goalReached, "Fundraising goal was reached");
        require(
            newFundraisingDeadline > fundraisingDeadline,
            "new fundraising deadline must be > old one"
        );
+       require(newFundraisingDeadline<fundExpiry "
    newFundraisingDeadline<fundExpiry");
        fundraisingDeadline = newFundraisingDeadline;
    }
```

[L3] Wrong require condition in `Daoo::refund`

Summary

The `Daoo::refund` function does not allow refund when `block.timestamp == fundraisingDeadline` .

Finding Description

In the `Daoo::refund` the requirement based on the `block.timestamp` does not allow refunds when `block.timestamp == fundraisingDeadline`.

```
function refund() external nonReentrant {
        require(!goalReached, "Fundraising goal was reached");
@>      require(
        block.timestamp > fundraisingDeadline,
            "Deadline not reached yet"
        );
        require(contributions[msg.sender] > 0, "No contributions to
           refund");

        uint256 contributedAmount = contributions[msg.sender];
        contributions[msg.sender] = 0;

        payable(msg.sender).transfer(contributedAmount);

        emit Refund(msg.sender, contributedAmount);
    }
```

However to be coherent with the `Daoo::contribute` and `Dao::receive` it should be `require(block.timestamp >= fundraisingDeadline, "Deadline not reached yet");`.

```
function contribute() public payable nonReentrant {
        require(!goalReached, "Goal already reached");
@>      require(block.timestamp < fundraisingDeadline, "Deadline hit");
        require(msg.value > 0, "Contribution must be greater than 0");
```

```
receive() external payable {
@>      if (!goalReached && block.timestamp < fundraisingDeadline) {
            contribute();
        }
    }
```

Impact Explanation

When `block.timestamp==fundraisingDeadline` users cannot do anything neither contributing nor refunding their ETH.

Likelihood Explanation

The likelihood is very low as `block.timestamp==fundraisingDeadline` lasts 1 second.

Recommendation

```
function refund() external nonReentrant {
        require(!goalReached, "Fundraising goal was reached");
-        require(
-        block.timestamp > fundraisingDeadline,
-            "Deadline not reached yet"
-        )
+       require(
+       block.timestamp >= fundraisingDeadline,
+           "Deadline not reached yet"
+        );
        require(contributions[msg.sender] > 0, "No contributions to
           refund");

        uint256 contributedAmount = contributions[msg.sender];
        contributions[msg.sender] = 0;

        payable(msg.sender).transfer(contributedAmount);

        emit Refund(msg.sender, contributedAmount);
    }
```

[I1] Could be useful to add a new `require` in `Dao::extendFundExpiry`

Summary

In case of `!fundraisingFinalized` the `Dao::extendFundExpiry` will revert

Finding Description

If the `owner` calls the `Dao::extendFundExpiry` when `!fundraisingFinalized` the function will revert because `liquidityLocker=address(0)`.

```
function extendFundExpiry(uint256 newFundExpiry) external onlyOwner {
        require(newFundExpiry > fundExpiry, "Must choose later fund
            expiry");
        fundExpiry = newFundExpiry;
@>      ILocker(liquidityLocker).extendFundExpiry(newFundExpiry);
    }
```

Indeed the variable `liquidityLocker` is changed by its default value only if `Daoo::finalizeFundraising` is called.

```
function finalizeFundraising(int24 initialTick, int24 upperTick)
    external {

      .
      .
      .

     emit TokenTransferredToLocker(tokenId, lockerAddress);

         // Initialize the locker
         ILocker(lockerAddress).initializer(tokenId);
         emit LockerInitialized(tokenId);

@>       liquidityLocker = lockerAddress;
         emit DebugLog("Finalize fundraising complete");
    }
```

Recommendation

```
function extendFundExpiry(uint256 newFundExpiry) external onlyOwner {
        require(newFundExpiry > fundExpiry, "Must choose later fund
            expiry");
+       require(fundraisingFinalized, "!fundraisingFinalized`);
        fundExpiry = newFundExpiry;
        ILocker(liquidityLocker).extendFundExpiry(newFundExpiry);
    }
```

[I2] Remove the secondToken variable as not used

Recommendation

Remove the `secondToken` variable as it is not used at all but only set.

```
    address public secondToken;
```

```
function setSecondToken(address _daoToken) external onlyOwner {
        require(_daoToken != address(0), "Invalid second token address"
            );
        require(secondToken == address(0), "DAO token already set");
        secondToken = _daoToken;
    }
```

[I3] Missing check in the `Dao::addToWhitelist` may lead to misalignment with the contibution `tiers`

Summary

Calling `Dao::addToWhitelist` could create a misalignment between `userLimits` and `tierslimits` in terms of any contribution already done.

Finding Description

Using `Dao::addToWhitelist` the trusted role may overwrite a whitelisted user and lead to the inconsistency `contributions[msg.sender] >= userLimit`.

Impact Explanation

Inconsistency between `max contribution tiers` and all the contributions done by single `whitelisted` users

Likelihood Explanation

Medium

Recommendation

```
for (uint256 i = 0; i < _addresses.length; i++) {
            require(_addresses[i] != address(0), "Invalid address");
            require(_tiers[i] != WhitelistTier.None, "Invalid tier");
+            uint256 userLimit = tierLimits[_tiers[i] ];
+            require(contributions[msg.sender] <= userLimit, "Invalid
    tier");

            userTiers[_addresses[i]] = _tiers[i];
```