# FactoryDAO contest
# Findings & Analysis Report

2022-08-02

## Table of contents

## Overview

### About C4

Code4rena (C4) is an open organization consisting of security researchers, auditors, developers, and individuals with domain expertise in smart contracts.

A C4 audit contest is an event in which community participants, referred to as Wardens, review, audit, or analyze smart contract logic in exchange for a bounty provided by sponsoring projects.

During the audit contest outlined in this document, C4 conducted an analysis of the FactoryDAO smart contract system written in Solidity. The audit contest took place between May 4—May 8 2022.

## Wardens

81 Wardens contributed reports to the FactoryDAO contest:

1. IllIllI
2. AuditsAreUS
3. kenzo
4. unforgiven
5. hyh
6. fatherOfBlocks
7. danb
8. WatchPug (jtp and ming)
9. pedroais
10. horsefacts
11. leastwood
12. reassor
13. hickuphh3
14. gzeon
15. Picodes
16. GimelSec (rayn and sces60107)
17. 0xf15ers (remora and twojoy)
18. PPrieditis
19. 0xYamiDancho
20. 0x52

21. scaraven
22. Dravee
23. defsec
24. robee
25. VAD37
26. rajatbeladiya
27. csanuragjain
28. sorrynotsorry
29. hubble (ksk2345 and shri4net)
30. shenwilly
31. eccentricexit
32. rfa
33. joestakey
34. oyc_109
35. MaratCerby
36. Ruhum
37. 0x1f8b
38. samruna
39. TerrierLover
40. ilan
41. berndartmueller
42. ellahi
43. hansfriese
44. 0xkatana
45. juicy
46. Funen
47. simon135
48. Hawkeye (0xwags and 0xmint)
49. delfin454000

50. ACai
51. ych18
52. throttle
53. kebabsec (okkothejawa and FlameHorizon)
54. AlleyCat
55. Bruhhh
56. cccz
57. CertoraInc (egjlmn1, OriDabush, ItayG, and shakedwinder)
58. 0xNazgul
59. z3s
60. Tomio
61. Waze
62. minhquanym
63. wuwe1
64. broccolirob
65. cryptphi
66. peritoflores
67. 0x1337
68. jayjonah8
69. mtz
70. p4st13r4 (0x69e8 and 0xb4bb4)
71. TrungOre

This contest was judged by Justin Goro.

Final report assembled by itsmetechjay.

## Summary

The C4 analysis yielded an aggregated total of 24 unique vulnerabilities. Of these vulnerabilities, 3 received a risk rating in the category of HIGH severity and 21 received a risk rating in the category of MEDIUM severity.

Additionally, C4 analysis included 43 reports detailing issues with a risk rating of LOW severity or non-critical. There were also 40 reports recommending gas optimizations.

All of the issues presented here are linked back to their original finding.

## Scope

The code under review can be found within the C4 FactoryDAO contest repository, and is composed of 10 smart contracts written in the Solidity programming language and includes 812 lines of Solidity code.

## Severity Criteria

C4 assesses the severity of disclosed vulnerabilities according to a methodology based on OWASP standards.

Vulnerabilities are divided into three primary risk categories: high, medium, and low/non-critical.

High-level considerations for vulnerabilities span the following key areas when conducting assessments:

- Malicious Input Handling
- Escalation of privileges
- Arithmetic
- Gas use

Further information regarding the severity criteria referenced throughout the submission review process, please refer to the documentation provided on the C4 website.

## High Risk Findings (3)

### [H-01] SpeedBumpPriceGate: Excess ether did not return to the user

*Submitted by cccz, also found by 0x52, 0xYamiDancho, csanuragjain, GimelSec, gzeon, hickuphh3, horsefacts, hyh, IllIllI, kenzo, leastwood, PPrieditis, reassor, unforgiven, WatchPug, and danb*

The `passThruGate` function of the `SpeedBumpPriceGate` contract is used to charge NFT purchase fees. Since the price of NFT will change due to the previous purchase, users are likely to send more ether than the actual purchase price in order to ensure that they can purchase NFT. However, the passThruGate function did not return the excess ether, which would cause asset loss to the user. Consider the following scenario:

1. An NFT is sold for 0.15 eth
2. User A believes that the value of the NFT is acceptable within 0.3 eth, considering that someone may buy the NFT before him, so user A transfers 0.3 eth to buy the NFT
3. When user A's transaction is executed, the price of the NFT is 0.15 eth, but since the contract does not return excess eth, user A actually spends 0.3 eth.

## Proof of Concept

[https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/SpeedBumpPriceGate.sol#L65-L82](https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/SpeedBumpPriceGate.sol#L65-L82)

## Recommended Mitigation Steps

```
-    function passThruGate(uint index, address) override external
+   function passThruGate(uint index, address payer) override ext
        uint price = getCost(index);
        require(msg.value >= price, 'Please send more ETH');

        // bump up the price
        Gate storage gate = gates[index];
        // multiply by the price increase factor
        gate.lastPrice = (price * gate.priceIncreaseFactor) / ga
        // move up the reference
        gate.lastPurchaseBlock = block.number;

        // pass thru the ether
        if (msg.value > 0) {
                // use .call so we can send to contracts, for exampl
-               (bool sent, bytes memory data) = gate.beneficiary.ca
+              (bool sent, bytes memory data) = gate.beneficiary.cal
                require(sent, 'ETH transfer failed');
            }
+       if (msg.value - price > 0){
+             (bool sent, bytes memory data) = payer.call{value: msg
+              require(sent, 'ETH transfer failed');}
        }
```

[illuzen (FactoryDAO) confirmed, but disagreed with severity](#)

[illuzen (FactoryDAO) resolved](#):

> https://github.com/code-423n4/2022-05-factorydao/pull/4

[Justin Goro (judge) commented](#):

> Maintaining severity as user funds are lost.

## [H-02] DoS: Blacklisted user may prevent `withdrawExcessRewards()`

*Submitted by AuditsAreUS*

https://github.com/code-423n4/2022-05-factorydao/blob/db415804c06143d8af6880bc4cda7222e5463c0e/contracts/PermissionlessBasicPoolFactory.sol#L242-L256

https://github.com/code-423n4/2022-05-factorydao/blob/db415804c06143d8af6880bc4cda7222e5463c0e/contracts/PermissionlessBasicPoolFactory.sol#L224-L234

### Impact

If one user becomes blacklisted or otherwise cannot be transferred funds in any of the rewards tokens or the deposit token then they will not be able to call `withdraw()` for that token.

The impact of one user not being able to call `withdraw()` is that the owner will now never be able to call `withdrawExcessRewards()` and therefore lock not only the users rewards and deposit but also and excess rewards attributed to the owner.

Thus, one malicious user may deliberately get them selves blacklisted to prevent the owner from claiming the final rewards. Since the attacker may do this with negligible balance in their `deposit()` this attack is very cheap.

## Proof of Concept

It is possible for `IERC20(pool.rewardTokens[i]).transfer(receipt.owner, transferAmount);` to fail for numerous reasons. Such as if a user has been blacklisted (in certain ERC20 tokens) or if a token is paused or there is an attack and the token is stuck.

This will prevent `withdraw()` from being called.

```
            for (uint i = 0; i < rewards.length; i++) {
                pool.rewardsWeiClaimed[i] += rewards[i];
                pool.rewardFunding[i] -= rewards[i];
                uint tax = (pool.taxPerCapita * rewards[i]) / 1000;
                uint transferAmount = rewards[i] - tax;
                taxes[poolId][i] += tax;
                success = success && IERC20(pool.rewardTokens[i]).tr
            }

            success = success && IERC20(pool.depositToken).transfer(
            require(success, 'Token transfer failed');
```

Since line 245 of `withdrawExcessRewards()` requires that `require(pool.totalDepositsWei == 0, 'Cannot withdraw until all deposits are withdrawn');` , if one single user is unable to withdraw then it is impossible for the owner to claim the excess rewards and they are forever stuck in the contract.

## Recommended Mitigation Steps

Consider allowing `withdrawExcessRewards()` to be called after a set period of time after the pool end if most users have withdrawn or some similar criteria.

illuzen (FactoryDAO) confirmed, disagreed with severity and commented:

> Kind of a duplicate, but I like this angle better. However, malicious token contracts are explicitly considered in the contract as an acceptable risk. Will probably just wrap transfers in a try catch

illuzen (FactoryDAO) resolved:

> https://github.com/code-423n4/2022-05-factorydao/pull/2

Justin Goro (judge) commented:

> Maintaining severity as deposit tokens at risk.

## [H-03] MerkleVesting withdrawal does not verify that tokens were transferred successfully

*Submitted by kenzo, also found by IllIllI*

Across the codebase, the protocol is usually checking that ERC20 transfers have succeeded by checking their return value. This check is missing in MerkleVesting's `withdraw` function.

### Impact

If for some reason the ERC20 transfer is temporarily failing, the user would totally lose his allocation and funds. All the state variables would already have been updated at this stage, so he can't call `withdraw` again. There is no way to withdraw these locked tokens.

### Proof of Concept

At the last point of `withdraw`, the function is sending the funds to the user, and does not check the return value - whether it has succeeded:

```
        IERC20(tree.tokenAddress).transfer(destination, currentW
```

Note that this is (nicely and rightfully) done after all the state variables have been updated. As the return value of the external call is not checked, if it has failed, the contract wouldn't know about it, and the function will finish "successfully".

### Recommended Mitigation Steps

As done throughout the rest of the protocol, add a check that verifies that the transfer has succeeded.

**illuzen (FactoryDAO) acknowledged, disagreed with severity and commented:**

> Debatable, since requiring successful transfer means we can't do non-standard tokens like USDT. Also, tokens could be malicious and simply lie about the success.

**Justin Goro (judge) commented:**

> Regarding the non standard tokens that don't return bools, the common approach to performing a low level call with

```
(bool success, _)  = address(token).call(//etc
```

> allows for transfers to be validated for USDT.

> Severity will stand because this function represents user funds.

## Medium Risk Findings (21)

### [M-01] `SpeedBumpPriceGate.sol#addGate()` Lack of input validation may casue div by 0 error

*Submitted by WatchPug*

https://github.com/code-423n4/2022-05-factorydao/blob/db415804c06143d8af6880bc4cda7222e5463c0e/contracts/SpeedBumpPriceGate.sol#L43

```
        gate.priceIncreaseDenominator = priceIncreaseDenominator;
```

If `priceIncreaseDenominator` is set to `0` when `addGate()`, in `passThruGate()` the tx will revert at L72 because of div by 0.

https://github.com/code-423n4/2022-05-factorydao/blob/db415804c06143d8af6880bc4cda7222e5463c0e/contracts/SpeedBumpPriceGate.sol#L71-L72

```
        // multiply by the price increase factor
        gate.lastPrice = (price * gate.priceIncreaseFactor) / gate.p
```

## Recommendation

Consider adding a check in `addGate()` to require `priceIncreaseDenominator > 0`.

illuzen (FactoryDAO) acknowledged, disagreed with severity and commented:

> This is fine, we just add another gate, redeploy the tree and only harm done is we lost some gas.

Justin Goro (judge) commented:

> While value is not leaked in this instance, this can cause functionality to be interrupted until fixed. Severity of issue will be maintained.

## [M-02] Malicious token reward could disable withdrawals

*Submitted by shenwilly, also found by 0xYamiDancho, hickuphh3, hubble, kenzo, and leastwood*

`PermissionlessBasicPoolFactory.withdraw` requires each reward token transfers to succeed before withdrawing the deposit. If one of the reward token is a malicious/pausable contract that reverts on transfer, unaware users that deposited into this pool will have their funds stuck in the contract.

### Recommended Mitigation Steps

Add an `emergencyWithdraw` function that ignores failed reward token transfers.

illuzen (FactoryDAO) confirmed and commented:

> This is explicitly acknowledged in the contract comments, malicious reward tokens render the pool malicious, there is no way to get around that, but the emergencyWithdraw idea is good

illuzen (FactoryDAO) commented:

> Technically duplicate, but mitigation is better here.

illuzen (FactoryDAO) resolved:

> https://github.com/code-423n4/2022-05-factorydao/pull/2

ksk2345 (warden) commented:

1. User Funds are at loss so severity of this issue is High. Please check the descriptions in the duplicate IssueIDs : #191, #145, and #106 (marked wrongly as dup of M13)
2. IssueId #192 and #246 are wrongly marked as dup of M02. The root cause is malicious rewardToken, but the impact is different than that of M02, and also the code fix for the impact will be different. M02 talks of impact on loss of user deposits and the fix will be in withdraw function. While these two Issues impact on globalBeneficiary not able to withdraw rewards, the fix will be in withdrawTaxes() function. Hence, these two issues needs to be separated into a New Medium Issue.

Justin Goro (judge) commented:

> @ksk2345 - I really do sympathize with your point of view. I think you made two good points. Here's the thing, though: When assessing these issues, it's very important to take sponsor's intent into account. That's what makes humans necessary in C4 (for now). Sufficiently powerful software can form a graph of vulnerabilities and draw inferences. Our job is to figure out if these vulnerabilities matter and to what extent.

> First to the broader point of linking duplicates, whether blocking users or taxes, the vector is via a malicious pool creator in the form of a reward token. The reason this matters is because the Factorydao pools can be permissionlessly created and completely ignored by users. They are analogous to Uniswap pools. Sure, someone could create a malicious token pair in Uniswap but since all pairs are opt-in and can be routed around, the use of this pair requires explicit consent from the end user.

> If we bear in mind that end user involvement is consensual and that this is communicated to the users and that nothing can be hidden on Ethereum then it follows that we can umbrella these issues not as malicious tokens or withdrawal and tax vulnerabilities but pool creators trying to game the code while relying on social engineering to funnel unsuspecting users through these channels (scam).

> **digression on duplicates**
> On a broader issue of duplicates, I notice a similar theme arising when duplicate labels are challenged so I'd just like to clarify how I grouped duplicates:

> For most of the duplicate disputes, it was "the issue reported the same but the fix was different." If the issue is invalid, however, then the fix is kind of irrelevant which is why I grouped those all as the same issue. If the issue is the same but the fixes different, then I grouped them if the fixes were qualitatively similar. If there was one amongst them that provided the best fix, this would be the original in the set.

## [M-03] safeTransferFrom is recommended instead of transfer (1)

*Submitted by MaratCerby, also found by berndartmueller, broccolirob, CertoraInc, cryptphi, danb, gzeon, horsefacts, hyh, joestakey, leastwood, throttle, VAD37, wuwe1, and z3s*

ERC20 standard allows transferF function of some contracts to return bool or return nothing.
Some tokens such as USDT return nothing.
This could lead to funds stuck in the contract without possibility to retrieve them.
Using safeTransferFrom of SafeERC20.sol is recommended instead.

### Proof of Concept

https://github.com/OpenZeppelin/openzeppelin-contracts/blob/4a9cc8b4918ef3736229a5cc5a310bdc17bf759f/contracts/token/ERC20/utils/SafeERC20.sol

illuzen (FactoryDAO) commented:

> We support ERC20 contracts, not SafeERC20. Contracts that do not conform to the standard are not supported.

illuzen (FactoryDAO) confirmed and resolved:

> https://github.com/code-423n4/2022-05-factorydao/pull/2

## [M-04] Merkle leaves are the same length as the parents that are hashed

*Submitted by AuditsAreUS*

https://github.com/code-423n4/2022-05-factorydao/blob/db415804c06143d8af6880bc4cda7222e5463c0e/contracts/MerkleLib.sol#L36-L42

https://github.com/code-423n4/2022-05-factorydao/blob/db415804c06143d8af6880bc4cda7222e5463c0e/contracts/MerkleDropFactory.sol#L94

### Impact

The size of a `leaf` is the same size of the parent data that is hashed, both are 64 bytes. As a result it is possible to have a hash collision between a leaf and any node in the tree. This allows for proofs to be repeated multiple times by taking subtrees as leaves.

Fraudulent proofs will disrupt the airdrop and transfer funds to invalid addresses.

### Proof of Concept

For example consider the following binary tree which has 4 leaves d,e,f,g

a->b
a->c

b->d
b->e

c->f
c->g

To calculate the parent hash for c it is `keccak(f || g)` and the parent hash for a is `keccak(b || c)` as seen in the `parentHash()` function below.

```
        function parentHash(bytes32 a, bytes32 b) public pure returr
            if (a < b) {
                return keccak256(abi.encode(a, b));
            } else {
                return keccak256(abi.encode(b, a));
            }
        }
```

A leaf is calculated as

```
    bytes32 leaf = keccak256(abi.encode(destination, value));
```

`abi.encode(address,uint)` will output 64 bytes. Since `abi.encode(bytes32,bytes32)` will also be 64 bytes it is possible to have a hash collision between a leaf and a parent node.

Taking the example above if we now set `destination = keccak(e || d)` and `value = keccak(f || g)` and provide the proof as an empty array since we are already at `a`, the root `[]`. This proof will verify for `destination` and `value` set to the hash of each child node.

This issue is rated as medium as there are some drawbacks to the attack that will make it challenging to pull off in practice. The first is that `destination` is a 20 bytes address and thus will require the node in the tree to have 12 leading zero bytes which may not occur. Second is the `value` is transferred to the user and so it is likely that the balance of the contract will not be sufficient for this transfer to succeed.

### Recommended Mitigation Steps

Consider using `leaf = keccak(abi.encodePacked(destination, value))` in `withdraw()` as this will reduce the size of the leaf data being hashed to 52 bytes.

Since `keccak256` prevents length extension attacks a different length of data to be hashed can be assumed to give different hashes and prevent a collision between a leaf and other nodes in the tree.

**illuzen (FactoryDAO) acknowledged and commented:**

> Valid. Interesting attack, but even if it is successful, it will transfer tokens to what is almost certainly an unusable address, making this both very unlikely (due to issues you mentioned) and of zero benefit to the attacker (griefing). 12 sequential zero bytes would occur approximately 1 out of every 16^12 = 281474976710656 nodes.

> I recommend bonus points for this one.

**Justin Goro (judge) commented:**

> Very impressive analysis.

## [M-05] `MerkleDropFactory.depositTokens()` does not require the tree to exist

*Submitted by AuditsAreUS*

The function `depositTokens()` does not first check to ensure that the `treeIndex` exists.

The impact is that we will attempt to transfer from the zero address to this address. If the transfer succeeds (which it currently does not since we use `IERC20.transferFrom()` ) then the `tokenBalance` of this index will be increased.

This will be an issue if the contract is updated to use OpenZeppelin's `safeTransferFrom()` function. This update may be necessary to support non-standard ERC20 tokens such as USDT.

If the update is made then `merkleTree.tokenAddress.safeTransferFrom(msg.sender, address(this), value), "ERC20 transfer failed");` will succeed if `merkleTree.tokenAddress = address(0)` since `safeTransferFrom()` succeeds against the zero address.

## Proof of Concept

There are no checks the `treeIndex` is valid.

```
    function depositTokens(uint treeIndex, uint value) public {
        // storage since we are editing
        MerkleTree storage merkleTree = merkleTrees[treeIndex];


        // bookkeeping to make sure trees don't share tokens
        merkleTree.tokenBalance += value;


        // transfer tokens, if this is a malicious token, then t
        // but it does not effect the other trees
        require(IERC20(merkleTree.tokenAddress).transferFrom(msg
        emit TokensDeposited(treeIndex, merkleTree.tokenAddress,
    }
```

## Recommended Mitigation Steps

Consider adding the check to ensure `0 < treeIndex <= numTrees` in `depositTokens()` .

illuzen (FactoryDAO) acknowledged, disagreed with severity and commented:

> Technically valid, but this harms no one but the caller, and incorrectly entering arguments is not in scope.

> File this under code style.

[illuzen (FactoryDAO) resolved]():

> [https://github.com/code-423n4/2022-05-factorydao/pull/3](https://github.com/code-423n4/2022-05-factorydao/pull/3)

[Justin Goro (judge) commented]():

> Maintaining severity as validating treeIndex isn't out of bounds seems within the appropriate expectations of input validation.

[HickupHH3 (warden) commented]():

> FYI, the call will not succeed because OZ's safeTransferFrom() will revert if target isn't an EOA. Solmate on the other hand will not.
> [https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/utils/SafeERC20.sol#L110](https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/utils/SafeERC20.sol#L110)
> [https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/Address.sol#L135](https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/Address.sol#L135)
> [https://github.com/Rari-Capital/solmate/blob/main/src/utils/SafeTransferLib.sol#L9](https://github.com/Rari-Capital/solmate/blob/main/src/utils/SafeTransferLib.sol#L9)

[Justin Goro (judge) commented]():

> @HickupHH3 - 'target' in openzeppelin's address library refers to the contract making the call. In other words, merkleTree.tokenAddress. So the call will succeed.

## [M-06] wrong out of range check

*Submitted by danb*

[https://github.com/code-423n4/2022-05-factorydao/blob/main/contracts/MerkleIdentity.sol#L124](https://github.com/code-423n4/2022-05-factorydao/blob/main/contracts/MerkleIdentity.sol#L124) https://github.com/code-423n4/2022-05-factorydao/blob/main/contracts/MerkleIdentity.sol#L98

## Vulnerability details

```
    require(merkleIndex <= numTrees, 'merkleIndex out of range');
```

This line checks that the index is valid.
However, the trees start at index 1, therefore index 0 should fail this check, but it doesn't.

## Recommended Mitigation Steps

change to:

```
    require(merkleIndex <= numTrees && merkleIndex > 0, 'merkleIndex
```

illuzen (FactoryDAO) acknowledged, disagreed with severity and commented:

> Technically valid, but the next line will revert, unless you think someone has the keccak pre-image of zero bytes and this pre-image is a set of valid arguments to this function...

illuzen (FactoryDAO) resolved:

> https://github.com/code-423n4/2022-05-factorydao/pull/4

## [M-07] getRewards() in PermissionlessBasicPoolFactory calculate wrong reward amount for receiptId==0

*Submitted by unforgiven*

In `getRewards()` of `PermissionlessBasicPoolFactory` contract, there is a check to see that receipt is initialized receipt, but the condition used by code will be true for `receiptId` equal `0`. because `receiptId==0` is not initilized for any pool and the value of `pools[poolId].receipts[0].id` will be `0` so the condition `receipt.id == receiptId` will be passed on `getRewards()`. Any function that depends on `getRewards()` to check that if `receptId` has deposited fund, can be fooled. right now this bug has no direct money loss, but this function doesn't work as it suppose too.

## Proof of Concept

This is `getRewards()` code:

```
        function getRewards(uint poolId, uint receiptId) public view
            Pool storage pool = pools[poolId];
            Receipt memory receipt = pool.receipts[receiptId];
            require(pool.id == poolId, 'Uninitialized pool');
            require(receipt.id == receiptId, 'Uninitialized receipt'
            uint nowish = block.timestamp;
            if (nowish > pool.endTime) {
                nowish = pool.endTime;
            }

            uint secondsDiff = nowish - receipt.timeDeposited;
            uint[] memory rewardsLocal = new uint[](pool.rewardsWeiP
            for (uint i = 0; i < pool.rewardsWeiPerSecondPerToken.le
                rewardsLocal[i] = (secondsDiff * pool.rewardsWeiPerS
            }
            return rewardsLocal;
        }
```

if the value of `receiptId` set as `0` then even so `receiptId==0` is not initialized but this line:

```
            require(receipt.id == receiptId, 'Uninitialized receipt'
```

will be passed, because, receipts start from number `1` and `pool.receipts[0]` will have zero value for his fields. This is the code in `deposit()` which is responsible for creating receipt objects.

```
            pool.totalDepositsWei += amount;
```

```
        pool.numReceipts++;

        Receipt storage receipt = pool.receipts[pool.numReceipts
        receipt.id = pool.numReceipts;
        receipt.amountDepositedWei = amount;
        receipt.timeDeposited = block.timestamp;
        receipt.owner = msg.sender;
```

as you can see `pool.numReceipts++` and `pool.receipts[pool.numReceipts]` increase `numReceipts` and use it as receipts index. so receipnts will start from index `1`.
This bug will cause that `getRewards(poolId, 0)` return `0` instead of reverting. any function that depend on reverting of `getRewards()` for uninitialized receipts can be excploited by sending `receipntId` as `0`. this function can be inside this contract or other contracts. ( `withdraw` use `getRewards` and we will see that we can create `WithdrawalOccurred` event for `receiptsId` as 0)

## Tools Used

VIM

## Recommended Mitigation Steps

If you want to start from index `1` then add this line too to ensure `receipntId` is not `0` too:

```
    require(receiptId > 0, 'Uninitialized receipt');
```

or we could check for uninitialized receipnts with `owner` field as non-zero.

illuzen (FactoryDAO) confirmed, disagreed with severity and commented:

> Technically valid, but it is a no-op, nothing bad happens either on withdraw or on getRewards. There are no functions that depend on getRewards reverting, but yes it would be cleaner if we do not allow 0 here.

## [M-08] A transfer that is not validated its result.

*Submitted by fatherOfBlocks*

When the transfer is made in the **withdraw()** function, it is not validated if the transfer was done correctly.

This could be a conflict since not being able to perform it would return a false and that case would not be handled, the most common is to revert.

### Recommended Mitigation Steps

The recommendation is to wrap the transfer with a require, as is done in **MerkleDropFactory.sol** for example.

**illuzen (FactoryDAO) acknowledged and commented:**

> Malicious or otherwise bad tokens are considered acceptable risks for this contract as long as they cannot interfere with other trees.

**illuzen (FactoryDAO) resolved:**

> https://github.com/code-423n4/2022-05-factorydao/pull/3

## [M-09] Rebasing tokens go to the pool owner, or remain locked in the various contracts

*Submitted by IllIllI*

Rebasing tokens are tokens that have each holder's `balanceof()` increase over time. Aave aTokens are an example of such tokens.

### Impact

Users expect that when they deposit tokens to a pool, that they get back all rewards earned, not just a flat rate. With the contracts of this project, deposited tokens will grow in value, but the value in excess of the pre-calculated `getMaximumRewards()` /deposited amounts go solely to the owner/creator, or will remain locked in the contract

## Proof of Concept

In the case of pools, the owner can withdraw the excess rebasing reward tokens by calling `withdrawExcessRewards()` , but is unable to withdraw excess deposited rebasing tokens. The Merkle-tree-related contracts have no way to withdraw any excess rebasing tokens.

All parts of the code assume that the value stated is the balance that is available to withdraw. It stores the values...

*(Note: see [submission](#) for full Proof of Concept)*

## Recommended Mitigation Steps

Provide a function for the pool owner to withdraw excess deposited tokens and repay any associated taxes. In the case of the Merkle trees though, pro rata share amounts need to be calculated and tracked and updated with every withdrawal, which will require drastic changes to the code, making it much more expensive.

**illuzen (FactoryDAO) acknowledged, disagreed with severity and commented:**

> Valid, but i think we will just not support rebasing tokens

**illuzen (FactoryDAO) resolved:**

> https://github.com/code-423n4/2022-05-factorydao/pull/3

**Justin Goro (judge) commented:**

> Severity maintained.

**0xleastwood (warden) commented:**

> Not supporting rebasing tokens is equivalent to not supporting fee-on-transfer tokens. Maybe we could group all non-standard ERC20 issues together?

**IllIllI000 (warden) commented:**

> They are slightly different: Fee-on-transfer tokens cause things to revert which will either prevent the token from being used, or will cause deposited funds to be locked. With rebasing tokens, a user misses out on new rewards, rather than losing deposited capital

[ksk2345 (warden) commented](#):

> If a sponsor mentions that they will not be supporting rebase tokens, then its judged as invalid.
> Refer : [https://github.com/code-423n4/2022-04-backed-findings/issues/105](https://github.com/code-423n4/2022-04-backed-findings/issues/105)
> I think we need a general consensus in C4 org rulebook how to treat this token and having some consistent judgement.

[Justin Goro (judge) commented](#):

> The sponsor acknowledged the validity and referenced this in a PR so it's sitting somewhere between acknowledged and confirmed. In other words, the sponsor is not averse to lending a helping hand to rebase tokens but will not explicitly encourage the use of them. This is similar to how a standard CFMM allows for rebase tokens to exist and function but not without side effects, similar to the PR which does not close all holes against rebase tokens.

## [M-10] Unbounded loop in `withdraw()` may cause rewards to be locked in the contract

*Submitted by IllIllI*

The `withdraw()` has an unbounded loop with external calls. If the gas costs of functions change between when deposits are made and when rewards are withdrawn, or if the gas cost of the deposit ( `transferFrom()` ) is less than the gas cost of the withdrawal ( `transfer()` ), then the `withdraw()` function may revert due to exceeding the block size gas limit.

### Proof of Concept

`transfer()` is an external call, and `rewards.length` has no maximum size:

```
File: contracts/PermissionlessBasicPoolFactory.sol   #1

224            for (uint i = 0; i < rewards.length; i++) {
225                pool.rewardsWeiClaimed[i] += rewards[i];
226                pool.rewardFunding[i] -= rewards[i];
227                uint tax = (pool.taxPerCapita * rewards[i]) /
228                uint transferAmount = rewards[i] - tax;
229                taxes[poolId][i] += tax;
230                success = success && IERC20(pool.rewardTokens[
231            }
```

[https://github.com/code-423n4/2022-05-factorydao/blob/db415804c06143d8af6880bc4cda7222e5463c0e/contracts/PermissionlessBasicPoolFactory.sol#L224-L231](https://github.com/code-423n4/2022-05-factorydao/blob/db415804c06143d8af6880bc4cda7222e5463c0e/contracts/PermissionlessBasicPoolFactory.sol#L224-L231)

### Recommended Mitigation Steps

Allow the specification of an offset and length to the `withdraw()` function, so that withdrawals can be broken up into smaller batches if required

[illuzen (FactoryDAO) confirmed]()

## [M-11] Pool owners can prevent the payment of taxes

*Submitted by IllIllI*

Pool owners can prevent taxes from being paid without impacting any other functionality

### Proof of Concept

By adding a custom reward token that always reverts for transfers to `globalBenericiary`, the owner can prevent taxes from being paid:

```
File: contracts/PermissionlessBasicPoolFactory.sol   #1

258        /// @notice Withdraw taxes from pool
259        /// @dev Anyone may call this, it just moves the taxes
260        /// @param poolId which pool are we talking about?
```

```
    function withdrawTaxes(uint poolId) external {
        Pool storage pool = pools[poolId];
        require(pool.id == poolId, 'Uninitialized pool');

        bool success = true;
        for (uint i = 0; i < pool.rewardTokens.length; i++
            uint tax = taxes[poolId][i];
            taxes[poolId][i] = 0;
            success = success && IERC20(pool.rewardTokens|
        }
        require(success, 'Token transfer failed');
    }
```

[https://github.com/code-423n4/2022-05-factorydao/blob/db415804c06143d8af6880bc4cda7222e5463c0e/contracts/PermissionlessBasicPoolFactory.sol#L258-L272](https://github.com/code-423n4/2022-05-factorydao/blob/db415804c06143d8af6880bc4cda7222e5463c0e/contracts/PermissionlessBasicPoolFactory.sol#L258-L272)

While the sponsor mentions that malicious tokens make the pool malicious, this particular issue has a simple fix outlined below in the mitigation section

## Recommended Mitigation Steps

Force taxes to be paid during `withdraw()`

**[illuzen (FactoryDAO) confirmed, disagreed with severity and commented]():**

> Valid, but adding another transfer to withdraw increases gas costs. And it's possible pool creator and token creator are not same party, so it's not clear the mitigation would be better.

> I think it's an acceptable risk. If someone wants to go to this level of trouble, they could just fork the contract and remove the fees.

**[illuzen (FactoryDAO) resolved]():**

> [https://github.com/code-423n4/2022-05-factorydao/pull/2](https://github.com/code-423n4/2022-05-factorydao/pull/2)

**[Justin Goro (judge) commented]():**

> Validating against tokens specifically written with if statements for this contract is not really something a developer can prevent. For instance, a token creator can cause their token to revert if the contract requesting transferFrom approval is a Uniswap router. This would prevent all trade of that token within Uniswap. But that's certainly not the failing of the Uniswap developers.

> However, for tokens in particular, it is recommended to not revert on bad implementations and so the issue will be treated as belonging to that camp of suggestions. For that reason, the risk status will remain 2.

## [M-12] Pool owners can prevent withdrawals of specific receipts

*Submitted by IllIllI*

Pool owners can prevent withdrawals of specific receipts without impacting any other functionality

### Proof of Concept

Reciepts are non-transferrable, so a malicious owner can monitor the blockchain for receipt creations, and inspect which account holds the receiptId. Next, by changing settings in a custom reward token that reverts for specific addresses, the owner can prevent that specific receipt owner from withdrawing:

```
File: contracts/PermissionlessBasicPoolFactory.sol   #1

230                 success = success && IERC20(pool.rewardTokens[
231             }
232
233             success = success && IERC20(pool.depositToken).tra
234             require(success, 'Token transfer failed');
```

[https://github.com/code-423n4/2022-05-factorydao/blob/db415804c06143d8af6880bc4cda7222e5463c0e/contracts/PermissionlessBasicPoolFactory.sol#L230-L234](https://github.com/code-423n4/2022-05-factorydao/blob/db415804c06143d8af6880bc4cda7222e5463c0e/contracts/PermissionlessBasicPoolFactory.sol#L230-L234)

While the sponsor mentions that malicious tokens make the pool malicious, this particular issue has a straight forward fix outlined below in the mitigation section

### Recommended Mitigation Steps

Rather than reverting the whole withdrawal if only one transfer fails, return a boolean of whether all withdrawals were successful, and allow `withdraw()` to be called multiple times, keeping track of what has been transferred and what hasn't

illuzen (FactoryDAO) confirmed, disagreed with severity and commented:

> Valid, similar to #124 mitigation seems risky, could provide emergencyWithdrawal function instead...

illuzen (FactoryDAO) resolved:

> https://github.com/code-423n4/2022-05-factorydao/pull/2

Justin Goro (judge) commented:

> While the example provided is not in scope for the developer to fix, the catch all provided by the sponsor is a good way to act as a last resort protection against loop griefing attacks.

## [M-13] amount requires to be updated to contract balance increase (1)

*Submitted by MaratCerby, also found by 0x1337, 0x52, 0xYamiDancho, AuditsAreUS, berndartmueller, cccz, CertoraInc, csanuragjain, defsec, Dravee, GimelSec, hickuphh3, horsefacts, hyh, IllIIIl, jayjonah8, kenzo, leastwood, mtz, p4st13r4, PPrieditis, reassor, Ruhum, throttle, TrungOre, VAD37, wuwe1, and ych18*

Every time transferFrom or transfer function in ERC20 standard is called there is a possibility that underlying smart contract did not transfer the exact amount entered.

It is required to find out contract balance increase/decrease after the transfer. This pattern also prevents from re-entrancy attack vector.

## Recommended Mitigation Steps

Recommended code:

```
    function fundPool(uint poolId) internal {
        Pool storage pool = pools[poolId];
        bool success = true;
        uint amount;
        for (uint i = 0; i < pool.rewardFunding.length; i++) {
            amount = getMaximumRewards(poolId, i);
            // transfer the tokens from pool-creator to this contrac


            uint256 balanceBefore = IERC20(pool.rewardTokens[i]).bal
            IERC20(pool.rewardTokens[i]).safeTransferFrom(msg.sender
            uint256 newAmount = IERC20(pool.rewardTokens[i]).balance
            success = success && newAmount == amount; // making sure

            // bookkeeping to make sure pools don't share tokens
            pool.rewardFunding[i] += amount;
        }
        require(success, 'Token deposits failed');
    }
```

**illuzen (FactoryDAO) confirmed, disagreed with severity and commented:**

> Re-entrance here would involve sending our contract tokens multiple times and creating multiple pools, not withdraw any funds. Malicious tokens could lie about balance as well, so the mitigation doesn't completely fix the issue.

> And malicious tokens are explicitly considered in the comments as acceptable. What is unacceptable is malicious pools harming other pools.

**illuzen (FactoryDAO) resolved:**

> https://github.com/code-423n4/2022-05-factorydao/pull/2

**Justin Goro (judge) decreased severity to Medium and commented:**

> Just a note on the reason for checking token balances before and after: not all tokens that report a difference between the balance and the amount are acting

> maliciously. In particular fee-on-transfer tokens.
> Reducing severity as this is a value leakage situation and because the sponsor has taken pains to emphasize the isolation of pools and the desire to not have to support all tokens.

## [M-14] Merkle-tree-related contracts vulnerable to cross-chain-replay attacks

*Submitted by IllIllI*

> Bank is a token vesting, airdrop and payroll tool. It uses merkle trees to massively scale token distributions with integrated vesting (time locks). The idea of this tool is that it allows DAOs to vest pre-sale participants, and future allocations of tokens (such as DAO treasury allocations) far into the future. These are important contracts since they need longevity and will secure large allocations of tokens.

https://github.com/code-423n4/2022-05-factorydao/blob/db415804c06143d8af6880bc4cda7222e5463c0e/README.md?plain=1#L28

Since these trees are long-lived, they need to be able to handle forks correctly. If someone generates an exchange address for their drops, that address may only be valid for that chain (e.g. exchange supports BTC but not BSV), and any funds sent to the unsupported chain are lost.

### Impact

If there's a fork, since anyone can call `withdraw()`, an attacker can monitor the blockchain for calls to `withdraw()`, and then make the same call with the same arguments on the other chain, which will send funds to the unsupported address.

### Proof of Concept

There are no EIP-712 protections in the encoding:

```
File: contracts/MerkleDropFactory.sol   #1
```

```
94          bytes32 leaf = keccak256(abi.encode(destination, va
```

[https://github.com/code-423n4/2022-05-factorydao/blob/db415804c06143d8af6880bc4cda7222e5463c0e/contracts/MerkleDropFactory.sol#L94](https://github.com/code-423n4/2022-05-factorydao/blob/db415804c06143d8af6880bc4cda7222e5463c0e/contracts/MerkleDropFactory.sol#L94)

```
File: contracts/MerkleVesting.sol   #2

109          bytes32 leaf = keccak256(abi.encode(destination, t
```

[https://github.com/code-423n4/2022-05-factorydao/blob/db415804c06143d8af6880bc4cda7222e5463c0e/contracts/MerkleVesting.sol#L109](https://github.com/code-423n4/2022-05-factorydao/blob/db415804c06143d8af6880bc4cda7222e5463c0e/contracts/MerkleVesting.sol#L109)

and anyone can trigger a withdrawal:

```
File: contracts/MerkleDropFactory.sol   #3

82       /// @dev Anyone may call this function for anyone else,
83       /// @dev who provides the proof and pays the gas, msg.s
```

[https://github.com/code-423n4/2022-05-factorydao/blob/db415804c06143d8af6880bc4cda7222e5463c0e/contracts/MerkleDropFactory.sol#L82-L83](https://github.com/code-423n4/2022-05-factorydao/blob/db415804c06143d8af6880bc4cda7222e5463c0e/contracts/MerkleDropFactory.sol#L82-L83)

## Recommended Mitigation Steps

Add EIP-712 protections and add a mechanism to allow tokens to be transferred to a different address using EIP-2612 `permit()`

[illuzen (FactoryDAO) acknowledged, disagreed with severity and commented](#):

> Sending funds to an unusable address on a chain that we didn't intend to be on doesn't seem in scope, but good thinking. In any case, the exchange will have the key for the address on the other chain since private keys are chain agnostic.

## [M-15] PermissionlessBasicPoolFactory's withdraw can become frozen on zero reward token transfers

*Submitted by hyh*

Reward tokens that do not allow for zero amount transfers can prevent user pool exit.

Now it is required that all reward amounts be successfully transferred to a receipt owner and the reward token amount isn't checked in the process.

If withdraw was called at the moment when some reward amount is zero (because either zero time passed or zero slope is set), the withdraw() will revert.

Say once such reward token is there (say with no malicious intent, as it's just a specifics of some valid tokens), user cannot withdraw immediately after deposit as no rewards accrued yet and this token transfer will revert the whole call even if it is one of the many.

As withdraw() the only way for a user to exit pool, her funds will be frozen within.

If slope is set to zero for such a token, either maliciously or mistakenly, the withdrawals are impossible for all the users.

As this is user fund freeze case with external assumptions, setting the severity to medium.

### Proof of Concept

Some ERC20 tokens do not allow for zero amount transfers:

https://github.com/d-xo/weird-erc20#revert-on-zero-value-transfers

withdraw() iterates across the set of reward tokens, and requires all transfers to go through:

https://github.com/code-423n4/2022-05-factorydao/blob/db415804c06143d8af6880bc4cda7222e5463c0e/contracts/PermissionlessBasicPoolFactory.sol#L230-L230

```
    success = success && IERC20(pool.rewardTokens[i]).transfer(recei
```

Once some is not ok, the whole call reverts. As it’s the only way for a user to exit the pool, her funds are frozen until non-zero reward is obtained.

It might never happen as rewardsWeiPerSecondPerToken is allowed to be zero:

https://github.com/code-423n4/2022-05-factorydao/blob/db415804c06143d8af6880bc4cda7222e5463c0e/contracts/PermissionlessBasicPoolFactory.sol#L92-L112

```
        function addPool (
            uint startTime,
            uint maxDeposit,
            uint[] memory rewardsWeiPerSecondPerToken,
            uint programLengthDays,
            address depositTokenAddress,
            address excessBeneficiary,
            address[] memory rewardTokenAddresses,
            bytes32 ipfsHash,
            bytes32 name
        ) external {
            Pool storage pool = pools[++numPools];
            pool.id = numPools;
            pool.rewardsWeiPerSecondPerToken = rewardsWeiPerSecondPe
            pool.startTime = startTime > block.timestamp ? startTime
            pool.endTime = pool.startTime + (programLengthDays * 1 c
            pool.depositToken = depositTokenAddress;
            pool.excessBeneficiary = excessBeneficiary;
            pool.taxPerCapita = globalTaxPerCapita;

            require(rewardsWeiPerSecondPerToken.length == rewardToke
```

This way, when one of the reward tokens doesn’t allow for zero transfers:

1. immediate withdraw after deposit is impossible
2. factory allows for creation of malicious or misconfigured pools by adding such a reward token with zero rewardsWeiPerSecondPerToken, making withdraw impossible for all users

## Recommended Mitigation Steps

Consider controlling for zero amounts in reward transfer cycle:

```
        for (uint i = 0; i < rewards.length; i++) {
+                if (rewards[i] > 0) {
                    pool.rewardsWeiClaimed[i] += rewards[i];
                    pool.rewardFunding[i] -= rewards[i];
                    uint tax = (pool.taxPerCapita * rewards[i])
                    uint transferAmount = rewards[i] - tax;
                    taxes[poolId][i] += tax;
                    success = success && IERC20(pool.rewardToker
+                }
        }
```

[https://github.com/code-423n4/2022-05-factorydao/blob/db415804c06143d8af6880bc4cda7222e5463c0e/contracts/PermissionlessBasicPoolFactory.sol#L224-L231](https://github.com/code-423n4/2022-05-factorydao/blob/db415804c06143d8af6880bc4cda7222e5463c0e/contracts/PermissionlessBasicPoolFactory.sol#L224-L231)

illuzen (FactoryDAO) marked as duplicate and commented:

> Duplicate of #108 (M-13)

Justin Goro (judge) commented:

> Unmarking duplicate as FOT and revert-on-zero tokens are different.
> Consider this issue implicitly acknowledged as sponsor has communicated that badly implemented ERC20 tokens are allowed so long as they respect pool isolation.
> However, this bug is still a useful boundary condition to consider and so it will not be marked as invalid.

## [M-16] ERC20 tokens with different decimals than 18 leads to loss of funds

*Submitted by reassor, also found by hyh, IllIIII, kenzo, leastwood, rajatbeladiya, VAD37, and ych18* https://github.com/code-423n4/2022-05-factorydao/blob/db415804c06143d8af6880bc4cda7222e5463c0e/contracts/PermissionlessBasicPoolFactory.sol#L169

https://github.com/code-423n4/2022-05-factorydao/blob/db415804c06143d8af6880bc4cda7222e5463c0e/contracts/PermissionlessBasicPoolFactory.sol#L282

### Impact

Contract `PermissionlessBasicPoolFactory` calculates rewards by using hardcoded value of decimals `18` (1e18) for ERC20 tokens. This leads to wrong rewards calculations and effectively loss of funds for all pools that will be using ERC20 tokens with different decimals than `18`. Example of such a token is USDC that has 6 decimals only.

### Proof of Concept

- https://github.com/code-423n4/2022-05-factorydao/blob/db415804c06143d8af6880bc4cda7222e5463c0e/contracts/PermissionlessBasicPoolFactory.sol#L169
- https://github.com/code-423n4/2022-05-factorydao/blob/db415804c06143d8af6880bc4cda7222e5463c0e/contracts/PermissionlessBasicPoolFactory.sol#L282

### Tools Used

Manual Review / VSCode

### Recommended Mitigation Steps

It is recommended to add support for different number of decimals than `18` by dynamically checking `decimals()` for the tokens that are part of the rewards calculations. Alternatively if such a support is not needed, new require statements should be added to `addPool` that will be checking that the number of decimals for all ERC20 tokens is `18`.

illuzen (FactoryDAO) confirmed

## [M-17] Owner of a pool may prevent any taxes being withdrawn

*Submitted by AuditsAreUS, also found by IllIllI*

It is possible for the owner of a pool to prevent any taxes being withdrawn by the `globalBeneficiary`. The impact is the taxed tokens will be permanently locked in the contract and `withdrawTaxes()` will not be callable for that `poolId`.

### Proof of Concept

The attack works by setting one of the `rewardTokenAddresses` to a malicious contract during `addPool()`. The malicious contract is set such that it will revert on the call `pool.rewardTokens[i]).transfer(globalBeneficiary, tax)` if an only if the `to` address is `globalBeneficiary.

The result of this attack is that if one reward transfer fails then entire `withdrawTaxes()` transaction will revert and no taxes can be claimed. However, the pool will function correctly for all other users.

```
    function withdrawTaxes(uint poolId) external {
        Pool storage pool = pools[poolId];
        require(pool.id == poolId, 'Uninitialized pool');


        bool success = true;
        for (uint i = 0; i < pool.rewardTokens.length; i++) {
            uint tax = taxes[poolId][i];
            taxes[poolId][i] = 0;
            success = success && IERC20(pool.rewardTokens[i]).tr
        }
        require(success, 'Token transfer failed');
    }
```

### Recommended Mitigation Steps

There are a few mitigations to this issue.

The first is for the `withdrawTaxes()` function to take both `poolId` and `rewardIndex` as a parameters to allowing the tax beneficiary to only withdraw from certain reward tokens in the pool. This would allow the beneficiary to withdraw from all reward tokens except malicious ones.

The second mitigation is to implement a `try-catch` condition around the withdrawal of reward tokens. In the catch statement re-instate the `taxes[poolId][i] = tax` if the transfer fails. Alternatively just skip the reward tokens if the transfer fails though this would be undesirable if a token is paused for some reason.

**illuzen (FactoryDAO) confirmed, disagreed with severity and commented:**

> Valid, will probably do `try-catch`.

**illuzen (FactoryDAO) resolved:**

> https://github.com/code-423n4/2022-05-factorydao/pull/2

**Justin Goro (judge) decreased severity to Medium and commented:**

> Downgraded: deposited funds not at risk but value leakage occurs.

## [M-18] DoS: Attacker may significantly increase the cost of `withdrawExcessRewards()` by creating a significant number of excess receipts

*Submitted by AuditsAreUS, also found by 0x52, 0xf15ers, and pedroais*

An attacker may cause a DoS attack on `withdrawExcessRewards()` by creating a excessive number of `receipts` with minimal value. Each of these receipts will need to be withdrawn before the owner can call `withdrawExcessRewards()`.

The impact is the owner would have to pay an unbounded amount of gas to `withdraw()` all the accounts and receive their excess funds.

### Proof of Concept

`withdrawExcessRewards()` has the requirement that `totalDepositsWei` for the pool is zero before the owner may call this function as seen on line 245.

```
            require(pool.totalDepositsWei == 0, 'Cannot withdraw unt
```

`pool.totalDepositsWei` is added to each time a user calls `deposit()`. It is increased by the amount the user deposits. There are no restrictions on the amount that may be deposited as a result a user may add 1 wei (or the smallest unit on any currency) which has negligible value.

The owner can force withdraw these accounts by calling `withdraw()` so long as `block.timestamp > pool.endTime`. They would be required to do this for each account that was created.

This could be a significant amount of gas costs, especially if the gas price has increased since the attacker originally made the deposits.

## Recommended Mitigation Steps

Consider adding a minimum deposit amount for each pool that can be configured by the pool owner.

Alternatively, allow the owner to call `withdrawExcessRewards()` given some other criteria such as

- A fix period of time (e.g. 1 month) has passed since the end of the auction; and
- 90% of the deposits have been withdrawn

These criteria can be customised as desired by the design team.

illuzen (FactoryDAO) confirmed and commented:

> Valid, will probably do minimum deposit.

## [M-19] Centralisation Risk: Owner may abuse the tax rate to claim 99.9% of pools

*Submitted by AuditsAreUS, also found by leastwood, pedroais, and reassor*

It is possible for the owner to increase the tax rate to 99.9% in `setGlobalTax()`.

The impact of this is that any future pools will be required to pay 99.9% of their rewards in tax to the `globalBeneficiary`.

It is possible for the `globalBeneficiary` to modify this and front-run any transactions in the mem-pool which call `addPool()`. These transactions will succeed and create pools with the 99.9% tax rate.

### Proof of Concept

The cap for the tax rate is 1000 = 100%.

```
    function setGlobalTax(uint newTaxPerCapita) external {
        require(msg.sender == globalBeneficiary, 'Only globalBer
        require(newTaxPerCapita < 1000, 'Tax too high');
        globalTaxPerCapita = newTaxPerCapita;
    }
```

### Recommended Mitigation Steps

It is recommended to put some reasonable upper bounds on the tax rate. Consider setting the upper bounds for the tax rate to 5%.

[illuzen (FactoryDAO) confirmed and resolved](#):

> https://github.com/code-423n4/2022-05-factorydao/pull/2

## [M-20] MerkleResistor: zero coinsPerSecond will brick tranche initialization and withdrawals

*Submitted by hickuphh3, also found by GimelSec, gzeon, and scaraven* https://github.com/code-423n4/2022-05-factorydao/blob/db415804c06143d8af6880bc4cda7222e5463c0e/contracts/MerkleResistor.sol#L259

https://github.com/code-423n4/2022-05-factorydao/blob/db415804c06143d8af6880bc4cda7222e5463c0e/contracts/MerkleResistor.sol#L264

## Details & Impact

It is possible for `coinsPerSecond` to be zero. In these cases, the `startTime` calculation

```
uint startTime = block.timestamp + vestingTime - (totalCoins / c
```

will revert from division by zero, preventing initialization, and by extension, withdrawals of vested tokens.

## Proof of Concept

We assume vesting time chosen is the maximum ( `tree.maxEndTime` ) so that `totalCoins = maxTotalPayments` . These examples showcase some possibilities for which the calculated `coinsPerSecond` can be zero.

### Example 1: High upfront percentage

- `pctUpFront = 99` (99% up front)
- `totalCoins = 10_000e6` (10k USDC)
- `vestingTime = 1 year`

```
uint coinsPerSecond = (totalCoins * (uint(100) - tree.pctUpFront
// 10_000e6 * (100 - 99) / (365 * 86400 * 100)
// = 0
```

### Example 2: Small reward amount / token decimals

- `pctUpFront = 0`
- `totalCoins = 100_000e2` (100k EURS)
- `vestingTime = 180 days`

```
uint coinsPerSecond = (totalCoins * (uint(100) - tree.pctUpFront
// 100_000e2 * 100 / (180 * 86400 * 100)
// = 0
```

### Recommended Mitigation Steps

Scale up `coinsPerSecond` by `PRECISION`, then scale down when executing withdrawals. While it isn't foolproof, the possibility of `coinsPerSecond` being zero is reduced significantly.

```
264
265 uint coinsPerSecond = (totalCoins * (uint(100) - tree.pctUpF
266
184
185 currentWithdrawal = (block.timestamp - tranche.lastWithdrawa
```

**illuzen (FactoryDAO) confirmed, disagreed with severity and commented:**

> Example 1 = 3, not 0

**Justin Goro (judge) decreased severity to Medium and commented:**

> Reducing severity because rewards are not staked user funds.

## [M-21] Verification should be leafed based and not address based

*Submitted by Picodes, also found by pedroais, and unforgiven*

https://github.com/code-423n4/2022-05-factorydao/blob/db415804c06143d8af6880bc4cda7222e5463c0e/contracts/Me rkleVesting.sol#L115

https://github.com/code-423n4/2022-05-factorydao/blob/db415804c06143d8af6880bc4cda7222e5463c0e/contracts/MerkleDropFactory.sol#L92

### Impact

Contracts should clarify what is the intended behavior for Merkle trees with multiple leafs with the same address.

### Recommended Mitigation Steps

There is 2 possible behaviors:

- either - what is currently done - you only authorize one claim per address, in which case the multiple leaf are here to give users a choice - for example you could use `MerkleVesting` to give users the choice between 2 sets of vesting parameters and have something close to `MerkleResistor`.
- either you use a mapping based on the leaf to store if a leaf has been claimed or not.

This behavior should be clarified in the comments at least, and made clear to merkle tree builders.

illuzen (FactoryDAO) confirmed, disagreed with severity and commented:

> This is covered in a comment on MerkleResistor:49, but we should put it elsewhere for clarity.

illuzen (FactoryDAO) resolved:

> https://github.com/code-423n4/2022-05-factorydao/pull/3

## Low Risk and Non-Critical Issues

For this contest, 43 reports were submitted by wardens detailing low risk and non-critical issues. The report highlighted below by **horsefacts** received the top score from the judge.

*The following wardens also submitted reports:* IllIllI, PPrieditis, robee, sorrynotsorry, defsec, reassor, rfa, MaratCerby, VAD37, berndartmueller, 0xYamiDancho, 0xf15ers, Ruhum, 0x1f8b, TerrierLover, ilan, ellahi, fatherOfBlocks, juicy, Dravee, eccentricexit, hickuphh3, hyh, GimelSec, joestakey, Picodes, Hawkeye, gzeon, hansfriese, oyc_109, 0xkatana, Funen, samruna, simon135, delfin454000, kebabsec, ACai, AlleyCat, Bruhhh, csanuragjain, rajatbeladiya, *and* throttle.

## [L-01] Missing parameter validations in `SpeedBumpPriceGate#addGate`

Callers of `addGate` can create price gates with a zero price floor (allowing users to claim free tokens), and zero `priceIncreaseDenominator` (causing price calculation to revert with a divide by zero error).

`SpeedBumpPriceGate#addGate`

```
    function addGate(uint priceFloor, uint priceDecay, uint pric
        // prefix operator increments then evaluates
        Gate storage gate = gates[++numGates];
        gate.priceFloor = priceFloor;
        gate.decayFactor = priceDecay;
        gate.priceIncreaseFactor = priceIncrease;
        gate.priceIncreaseDenominator = priceIncreaseDenominator
        gate.beneficiary = beneficiary;
    }
```

Suggestion: Validate that `priceFloor` and `priceIncreaseDenominator` are nonzero.

```
    function addGate(uint priceFloor, uint priceDecay, uint pric
        require(priceFloor != 0, "Price floor must be nonzero");
        require(priceIncreaseDenominator != 0, "Denominator must
        // prefix operator increments then evaluates
        Gate storage gate = gates[++numGates];
        gate.priceFloor = priceFloor;
        gate.decayFactor = priceDecay;
        gate.priceIncreaseFactor = priceIncrease;
        gate.priceIncreaseDenominator = priceIncreaseDenominator
        gate.beneficiary = beneficiary;
    }
```

## [L-02] `VoterID` token can be minted to the zero address

`VoterID` tokens can be minted to the zero address in `VoterID#createIdentityFor`.

[VoterID#createIdentityFor]

```
    function createIdentityFor(address thisOwner, uint thisToken
        require(msg.sender == _minter, 'Only minter may create i
        require(owners[thisToken] == address(0), 'Token already

        // for getTokenByIndex below, 0 based index so we do it
        allTokens[numIdentities] = thisToken;

        // increment the number of identities
        numIdentities = numIdentities + 1;

        // two way mapping for enumeration
        ownershipMapIndexToToken[thisOwner][balances[thisOwner]]
        ownershipMapTokenToIndex[thisOwner][thisToken] = balance

        // set owner of new token
        owners[thisToken] = thisOwner;
        // increment balances for owner
        balances[thisOwner] = balances[thisOwner] + 1;
        uriMap[thisToken] = uri;
        emit Transfer(address(0), thisOwner, thisToken);
        emit IdentityCreated(thisOwner, thisToken);
    }
```

Suggestion: validate `thisOwner` in `createIdentityFor`:

```
    function createIdentityFor(address thisOwner, uint thisToken
        require(msg.sender == _minter, 'Only minter may create i
        require(owners[thisToken] == address(0), 'Token already
        require(thisOwner != address(0), 'ERC721: mint to the ze

        // for getTokenByIndex below, 0 based index so we do it
        allTokens[numIdentities] = thisToken;

        // increment the number of identities
        numIdentities = numIdentities + 1;

        // two way mapping for enumeration
        ownershipMapIndexToToken[thisOwner][balances[thisOwner]]
        ownershipMapTokenToIndex[thisOwner][thisToken] = balance

        // set owner of new token
        owners[thisToken] = thisOwner;
        // increment balances for owner
        balances[thisOwner] = balances[thisOwner] + 1;
        uriMap[thisToken] = uri;
        emit Transfer(address(0), thisOwner, thisToken);
        emit IdentityCreated(thisOwner, thisToken);
    }
```

## [L-03] `VoterID` token can be minted to non-ERC721 receivers

`VoterID` tokens can be minted to non-ERC721 receivers in `VoterID#createIdentityFor`.

[`VoterID#createIdentityFor`]

```
    function createIdentityFor(address thisOwner, uint thisToker
        require(msg.sender == _minter, 'Only minter may create i
        require(owners[thisToken] == address(0), 'Token already

        // for getTokenByIndex below, 0 based index so we do it
        allTokens[numIdentities] = thisToken;

        // increment the number of identities
        numIdentities = numIdentities + 1;

        // two way mapping for enumeration
        ownershipMapIndexToToken[thisOwner][balances[thisOwner]]
        ownershipMapTokenToIndex[thisOwner][thisToken] = balance

        // set owner of new token
        owners[thisToken] = thisOwner;
        // increment balances for owner
        balances[thisOwner] = balances[thisOwner] + 1;
        uriMap[thisToken] = uri;
        emit Transfer(address(0), thisOwner, thisToken);
        emit IdentityCreated(thisOwner, thisToken);
    }
```

Suggestion: check `checkOnERC721Received` in `createIdentityFor`. This callback introduces a reentrancy vector, so take care to ensure callers of `createIdentityFor` use a reentrancy guard or follow checks-effects-interactions:

```
    function createIdentityFor(address thisOwner, uint thisToker
        require(msg.sender == _minter, 'Only minter may create i
        require(owners[thisToken] == address(0), 'Token already
        require(thisOwner != address(0), 'ERC721: mint to the ze

        // for getTokenByIndex below, 0 based index so we do it
        allTokens[numIdentities] = thisToken;

        // increment the number of identities
        numIdentities = numIdentities + 1;

        // two way mapping for enumeration
        ownershipMapIndexToToken[thisOwner][balances[thisOwner]]
        ownershipMapTokenToIndex[thisOwner][thisToken] = balance

        // set owner of new token
        owners[thisToken] = thisOwner;
        // increment balances for owner
        balances[thisOwner] = balances[thisOwner] + 1;
        uriMap[thisToken] = uri;

        require(
            checkOnERC721Received(address(0), thisOwner, thisTok
            "Identity: transfer to non ERC721Receiver implemente
        );
        emit Transfer(address(0), thisOwner, thisToken);
        emit IdentityCreated(thisOwner, thisToken);
    }
```

## [L-04] Prefer `safeTransfer` and `safeTransferFrom` for ERC20 token transfers

Consider using OpenZeppelin's `SafeERC20` library to handle edge cases in ERC20 token transfers. This prevents accidentally forgetting to check the return value, like the example in `MerkleVesting#withdraw`.

Potential changes:

- `PermissionlessBasicPoolFactory.sol#L144`
- `PermissionlessBasicPoolFactory.sol#L198`
- `PermissionlessBasicPoolFactory.sol#L230`
- `MerkleVesting.sol#L89`
- `MerkleResistor.sol#L121`
- `MerkleResistor.sol#L204`
- `MerkleDropFactory.sol#L77`
- `MerkleDropFactory.sol#L107`

## [L-05] Replace inline assembly with `account.code.length`

`<address>.code.length` can be used in Solidity >= 0.8.0 to access an account's code size and check if it is a contract without inline assembly.

`VoterID#isContract`

```
    function isContract(address account) internal view returns

        uint256 size;
        // solhint-disable-next-line no-inline-assembly
        assembly { size := extcodesize(account) }
        return size > 0;
    }
```

Suggestion:

```
        function isContract(address account) internal view returns
            return account.code.length != 0;
        }
```

## [L-06] `VoterID#transferFrom` does not distinguish nonexistent tokens from unapproved transfers

Unlike other common ERC721 implementations, `VoterID` does not distinguish an attempt to transfer a nonexistent token from an unapproved transfer:

[`VoterId#transferFrom`]

```
        function transferFrom(address from, address to, uint256 toke
            require(isApproved(msg.sender, tokenId), 'Identity: Unap
            transfer(from, to, tokenId);
        }
```

Consider checking that a token exists in `isApproved` to distinguish attempts to transfer nonexistint tokens. (See OpenZeppelin [`ERC721#_isApprovedOrOwner`] for an example).

## [N-01] Prefer two-step ownership transfers

If the `owner` of `VoterID` accidentally transfers ownership to an incorrect address, protected functions may become permanently inaccessible.

[`VoterID.sol#L151-L155`]

```
        function setOwner(address newOwner) external ownerOnly {
            address oldOwner = _owner_;
            _owner_ = newOwner;
            emit OwnerUpdated(oldOwner, newOwner);
        }
```

Suggestion: handle ownership transfers with two steps and two transactions. First, allow the current owner to propose a new owner address. Second, allow the proposed owner (and only the proposed owner) to accept ownership, and update the contract owner internally.

## [N-02] `balanceOf` does not revert on zero address query

According to the ERC721 spec and the natspec comment in the code, `VoterID#balanceOf` should revert when called with the zero address, but it does not:

[VoterID.sol#L168-L175]

```
    /// @notice Count all NFTs assigned to an owner
    /// @dev NFTs assigned to the zero address are considered ir
    ///  function throws for queries about the zero address.
    /// @param _address An address for whom to query the balance
    /// @return The number of NFTs owned by `owner`, possibly ze
    function balanceOf(address _address) external view returns (
        return balances[_address];
    }
```

Suggestion: Validate that `_address` is not `address(0)` in `balanceOf`:

```
    /// @notice Count all NFTs assigned to an owner
    /// @dev NFTs assigned to the zero address are considered ir
    ///  function throws for queries about the zero address.
    /// @param _address An address for whom to query the balance
    /// @return The number of NFTs owned by `owner`, possibly ze
    function balanceOf(address _address) external view returns (
        require(_address != address(0), "ERC721: balance query f
        return balances[_address];
    }
```

## [N-03] Move `require` check to top of function

The `require` check in `PermissionlessBasicPoolFactory#addPool` comes after several state changes. Consider moving it to the top of the function to follow the checks-effects-interactions pattern.

PermissionlessBasicPoolFactory.sol#L112

```
            require(rewardsWeiPerSecondPerToken.length == rewardToke
```

## [N-04] Emit events from privileged operations

Consider adding events to protected functions that change contract state. This enables you to monitor off chain for suspicious activity, and allows end users to observe and trust changes to these parameters.

- VoterId#setTokenURI
- MerkleIdentity#setManagement
- MerkleIdentity#setTreeAdder
- MerkleIdentity#setIpfsHash

## [N-05] Incomplete natspec comment

The `@notice` natspec comment on `VoterID` is incomplete.

## Gas Optimizations

For this contest, 40 reports were submitted by wardens detailing gas optimizations. The report highlighted below by IllIllI received the top score from the judge.

*The following wardens also submitted reports:* Dravee, defsec, gzeon, eccentricexit, 0xNazgul, 0xYamiDancho, joestakey, oyc_109, reassor, robee, samruna, 0xf15ers, 0x1f8b, Tomio, 0xkatana, hansfriese, rfa, Funen, TerrierLover, ilan, simon135, CertoraInc, Waze, ellahi, minhquanym, fatherOfBlocks, z3s, delfin454000, GimelSec, horsefacts, juicy, rajatbeladiya, PPrieditis, csanuragjain, Hawkeye, VAD37, Picodes, Ruhum, *and* ACai.

| | Title | Instances |
|---|---|---|
| 1 | `for` -loops should be broken out of earlier | 1 |

| | Title | Instances |
|---|---|---|
| 2 | Multiple `address` mappings can be combined into a single `mapping` of an `address` to a `struct`, where appropriate | 3 |
| 3 | State variables only set in the constructor should be declared `immutable` | 5 |
| 4 | Using `calldata` instead of `memory` for read-only arguments in `external` functions saves gas | 11 |
| 5 | State variables should be cached in stack variables rather than re-reading them from storage | 28 |
| 6 | `<x> += <y>` costs more gas than `<x> = <x> + <y>` for state variables | 1 |
| 7 | `internal` functions only called once can be inlined to save gas | 3 |
| 8 | Add `unchecked {}` for subtractions where the operands cannot underflow because of a previous `require()` | 2 |
| 9 | `<array>.length` should not be looked up in every loop of a `for`-loop | 7 |
| 10 | `++i` / `i++` should be `unchecked{++i}` / `unchecked{++i}` when it is not possible for them to overflow, as is the case when used in `for`- and `while`-loops | 6 |
| 11 | `require()` / `revert()` strings longer than 32 bytes cost extra gas | 19 |
| 12 | Not using the named return variables when a function returns, wastes deployment gas | 5 |
| 13 | Remove unused local variable | 2 |
| 14 | Using `bool`s for storage incurs overhead | 4 |
| 15 | `public` library function should be made `private` / `internal` | 1 |
| 16 | Move `if-else` to inside function call to save deployment gas | 1 |
| 17 | Use a more recent version of solidity | 12 |
| 18 | It costs more gas to initialize variables to zero than to let the default of zero be applied | 9 |
| 19 | `++i` costs less gas than `++i`, especially when it's used in `for`-loops (`--i` / `i--` too) | 7 |
| 20 | Using `private` rather than `public` for constants, saves gas | 1 |
| 21 | Duplicated `require()` / `revert()` checks should be refactored to a modifier or function | 2 |

|  | Title | Instances |
|---|---|---|
| 22 | Stack variable used as a cheaper cache for a state variable is only used once | 1 |
| 23 | `require()` or `revert()` statements that check input arguments should be at the top of the function | 3 |
| 24 | Use custom errors rather than `revert()` / `require()` strings to save deployment gas | 67 |
| 25 | `public` functions not called by the contract should be declared `external` instead | 8 |

Total: 209 instances over 25 classes

## [1] `for` -loops should be broken out of earlier

If it's known that the function will revert after the `for` -loop completes, `break` should be used to end the loop early

```
File: contracts/PermissionlessBasicPoolFactory.sol   #1

141          for (uint i = 0; i < pool.rewardFunding.length; i++
142              amount = getMaximumRewards(poolId, i);
143              // transfer the tokens from pool-creator to thi
144              success = success && IERC20(pool.rewardTokens[i
```

https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/PermissionlessBasicPoolFactory.sol#L141-L144

## [2] Multiple `address` mappings can be combined into a single `mapping` of an `address` to a `struct` , where appropriate

Saves a storage slot for the mapping. Depending on the circumstances and sizes of types, can avoid a Gsset (20000 gas) per mapping combined. Reads and subsequent writes can also be cheaper when a function requires both values and they both fit in the same storage slot

```
File: /contracts/VoterID.sol   #1

18       mapping (address => uint) public balances;
19
20       // Mapping from owner to operator approvals
21       mapping (address => mapping (address => bool)) public c
```

[https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/VoterID.sol#L18-L21](https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/VoterID.sol#L18-L21)

```
File: /contracts/VoterID.sol   #2

28       mapping (address => mapping (uint => uint)) public owne
29       mapping (address => mapping (uint => uint)) public owne
```

[https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/VoterID.sol#L28-L29](https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/VoterID.sol#L28-L29)

```
File: contracts/PermissionlessBasicPoolFactory.sol   #3

56      // pools[poolId] = poolStruct
57      mapping (uint => Pool) public pools;
58      // metadatas[poolId] = metadataStruct
59      mapping (uint => Metadata) public metadatas;
60      // taxes[poolId] = taxesCollected[rewardIndex]
61      mapping (uint => uint[]) public taxes;
```

[https://github.com/code-423n4/2022-05-factorydao/blob/db415804c06143d8af6880bc4cda7222e5463c0e/contracts/PermissionlessBasicPoolFactory.sol#L56-L61](https://github.com/code-423n4/2022-05-factorydao/blob/db415804c06143d8af6880bc4cda7222e5463c0e/contracts/PermissionlessBasicPoolFactory.sol#L56-L61)

## [3] State variables only set in the constructor should be declared `immutable`

Avoids a Gsset (20000 gas) in the constructor, and replaces each Gwarmacces (100 gas) with a `PUSH32` (3 gas).

```
File: /contracts/MerkleEligibility.sol   #1

16       address public gateMaster;
```

https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/MerkleEligibility.sol#L16

```
File: /contracts/PermissionlessBasicPoolFactory.sol   #2

51       address public globalBeneficiary;
```

https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/PermissionlessBasicPoolFactory.sol#L51

```
File: /contracts/VoterID.sol   #3

65       string _name;
```

https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/VoterID.sol#L65

```
File: /contracts/VoterID.sol   #4

66       string _symbol;
```

https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/VoterID.sol#L66

```
File: /contracts/VoterID.sol   #5

74       address public _minter;
```

https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/VoterID.sol#L74

## [4] Using `calldata` instead of `memory` for read-only arguments in `external` functions saves gas

When a function with a `memory` array is called externally, the `abi.decode()` step has to use a for-loop to copy each index of the `calldata` to the `memory` index. Each iteration of this for-loop costs at least 60 gas (i.e. `60 * <mem_array>.length`). Using `calldata` directly, obliviates the need for such a loop in the contract code and runtime execution. Structs have the same overhead as an array of length one

```
File: /contracts/interfaces/IVoterID.sol   #1

12       function createIdentityFor(address newId, uint tokenId,
```

https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/interfaces/IVoterID.sol#L12

```
File: /contracts/MerkleEligibility.sol   #2

85       function passThruGate(uint index, address recipient, by
```

https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/MerkleEligibility.sol#L85

```
File: /contracts/MerkleIdentity.sol   #3
```

```
116       function withdraw(uint merkleIndex, uint tokenId, stri
```

[https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/MerkleIdentity.sol#L116](https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/MerkleIdentity.sol#L116)

```
File: /contracts/MerkleIdentity.sol    #4

116       function withdraw(uint merkleIndex, uint tokenId, stri
```

[https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/MerkleIdentity.sol#L116](https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/MerkleIdentity.sol#L116)

```
File: /contracts/MerkleIdentity.sol    #5

116       function withdraw(uint merkleIndex, uint tokenId, stri
```

[https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/MerkleIdentity.sol#L116](https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/MerkleIdentity.sol#L116)

```
File: /contracts/MerkleResistor.sol    #6

134       function initialize(uint treeIndex, address destinatic
```

[https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/MerkleResistor.sol#L134](https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/MerkleResistor.sol#L134)

```
File: /contracts/MerkleVesting.sol    #7

104       function initialize(uint treeIndex, address destinatic
```

[https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/MerkleVesting.sol#L104](https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/MerkleVesting.sol#L104)

```
File: /contracts/PermissionlessBasicPoolFactory.sol   #8

95          uint[] memory rewardsWeiPerSecondPerToken,
```

[https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/PermissionlessBasicPoolFactory.sol#L95](https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/PermissionlessBasicPoolFactory.sol#L95)

```
File: /contracts/PermissionlessBasicPoolFactory.sol   #9

99          address[] memory rewardTokenAddresses,
```

[https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/PermissionlessBasicPoolFactory.sol#L99](https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/PermissionlessBasicPoolFactory.sol#L99)

```
File: /contracts/VoterID.sol   #10

162       function setTokenURI(uint token, string memory uri) e
```

[https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/VoterID.sol#L162](https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/VoterID.sol#L162)

```
File: /interfaces/IVoterID.sol   #11

12       function createIdentityFor(address newId, uint tokenId,
```

[https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/interfaces/IVoterID.sol#L12](https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/interfaces/IVoterID.sol#L12)

## [5] State variables should be cached in stack variables rather than re-reading them from storage

The instances below point to the second+ access of a state variable within a function. Caching will replace each Gwarmaccess (100 gas) with a much cheaper stack read. Less obvious fixes/optimizations include having local storage variables of mappings within state variable mappings or mappings within state variable structs, having local storage variables of structs within mappings, having local memory caches of state variable structs, or having local caches of state variable contracts/addresses.

See original submission for details.

## [6] `<x> += <y>` costs more gas than `<x> = <x> + <y>` for state variables

```
File: /contracts/MerkleEligibility.sol   #1

47            numGates += 1;
```

https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/MerkleEligibility.sol#L47

## [7] `internal` functions only called once can be inlined to save gas

Not inlining costs 20 to 40 gas because of two extra `JUMP` instructions and additional stack operations needed for function calls.

```
File: /contracts/PermissionlessBasicPoolFactory.sol   #1

137        function fundPool(uint poolId) internal {
```

https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/PermissionlessBasicPoolFactory.sol#L137

```
File: /contracts/VoterID.sol    #2

304        function transfer(address from, address to, uint256 to
```

https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/VoterID.sol#L304

```
File: /contracts/VoterID.sol    #3

343        function isContract(address account) internal view ret
```

https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/VoterID.sol#L343

## [8] Add `unchecked {}` for subtractions where the operands cannot underflow because of a previous `require()`

`require(a <= b); x = b - a` => `require(a <= b); unchecked { x = b - a }`

```
File: /contracts/MerkleVesting.sol    #1

157                currentWithdrawal = (block.timestamp - tranche
```

https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/MerkleVesting.sol#L157

```
File: /contracts/PermissionlessBasicPoolFactory.sol    #2
```

```
187                 amount = pool.maximumDepositWei - pool.totalDe
```

https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/PermissionlessBasicPoolFactory.sol#L187

## [9] `<array>.length` should not be looked up in every loop of a `for` -loop

The overheads outlined below are *PER LOOP*, excluding the first loop

- storage arrays incur a Gwarmaccess (100 gas)
- memory arrays use `MLOAD` (3 gas)
- calldata arrays use `CALLDATALOAD` (3 gas)

Caching the length changes each of these to a `DUP<N>` (3 gas), and gets rid of the extra `DUP<N>` needed to store the stack offset

```
File: /contracts/MerkleLib.sol   #1

22            for (uint i = 0; i < proof.length; i += 1) {
```

https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/MerkleLib.sol#L22

```
File: /contracts/PermissionlessBasicPoolFactory.sol   #2

115            for (uint i = 0; i < rewardTokenAddresses.length;
```

https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/PermissionlessBasicPoolFactory.sol#L115

```
File: /contracts/PermissionlessBasicPoolFactory.sol    #3

141            for (uint i = 0; i < pool.rewardFunding.length; i+
```

[https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/PermissionlessBasicPoolFactory.sol#L141](https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/PermissionlessBasicPoolFactory.sol#L141)

```
File: /contracts/PermissionlessBasicPoolFactory.sol    #4

168            for (uint i = 0; i < pool.rewardsWeiPerSecondPerT
```

[https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/PermissionlessBasicPoolFactory.sol#L168](https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/PermissionlessBasicPoolFactory.sol#L168)

```
File: /contracts/PermissionlessBasicPoolFactory.sol    #5

224            for (uint i = 0; i < rewards.length; i++) {
```

[https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/PermissionlessBasicPoolFactory.sol#L224](https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/PermissionlessBasicPoolFactory.sol#L224)

```
File: /contracts/PermissionlessBasicPoolFactory.sol    #6

249            for (uint i = 0; i < pool.rewardTokens.length; i+
```

[https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/PermissionlessBasicPoolFactory.sol#L249](https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/PermissionlessBasicPoolFactory.sol#L249)

```
File: /contracts/PermissionlessBasicPoolFactory.sol    #7
```

```
266           for (uint i = 0; i < pool.rewardTokens.length; i++
```

[https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/PermissionlessBasicPoolFactory.sol#L266](https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/PermissionlessBasicPoolFactory.sol#L266)

## [10] `++i` / `i++` should be `unchecked{++i}` / `unchecked{++i}` when it is not possible for them to overflow, as is the case when used in `for` - and `while` -loops

The `unchecked` keyword is new in solidity version 0.8.0, so this only applies to that version or higher, which these instances are. This saves 30-40 gas ***PER LOOP***

```
File: /contracts/PermissionlessBasicPoolFactory.sol   #1

115           for (uint i = 0; i < rewardTokenAddresses.length;
```

[https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/PermissionlessBasicPoolFactory.sol#L115](https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/PermissionlessBasicPoolFactory.sol#L115)

```
File: /contracts/PermissionlessBasicPoolFactory.sol   #2

141           for (uint i = 0; i < pool.rewardFunding.length; i+
```

[https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/PermissionlessBasicPoolFactory.sol#L141](https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/PermissionlessBasicPoolFactory.sol#L141)

```
File: /contracts/PermissionlessBasicPoolFactory.sol   #3

168           for (uint i = 0; i < pool.rewardsWeiPerSecondPerTo
```

https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/Per missionlessBasicPoolFactory.sol#L168

```
File: /contracts/PermissionlessBasicPoolFactory.sol    #4

224           for (uint i = 0; i < rewards.length; i++) {
```

https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/PermissionlessBasicPoolFactory.sol#L224

```
File: /contracts/PermissionlessBasicPoolFactory.sol    #5

249           for (uint i = 0; i < pool.rewardTokens.length; i+
```

https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/PermissionlessBasicPoolFactory.sol#L249

```
File: /contracts/PermissionlessBasicPoolFactory.sol    #6

266           for (uint i = 0; i < pool.rewardTokens.length; i+
```

https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/PermissionlessBasicPoolFactory.sol#L266

## [11] `require()` / `revert()` strings longer than 32 bytes cost extra gas

See original submission for full details.

## [12] Not using the named return variables when a function returns, wastes deployment gas

```
File: /contracts/FixedPricePassThruGate.sol    #1

40           return gate.ethCost;
```

[https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/FixedPricePassThruGate.sol#L40](https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/FixedPricePassThruGate.sol#L40)

```
File: /contracts/MerkleEligibility.sol    #2

50           return numGates;
```

[https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/MerkleEligibility.sol#L50](https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/MerkleEligibility.sol#L50)

```
File: /contracts/MerkleEligibility.sol    #3

77           return countValid && gate.totalWithdrawals < gate.m
```

[https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/MerkleEligibility.sol#L77](https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/MerkleEligibility.sol#L77)

```
File: /contracts/SpeedBumpPriceGate.sol    #4

56               return gate.priceFloor;
```

[https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/SpeedBumpPriceGate.sol#L56](https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/SpeedBumpPriceGate.sol#L56)

```
File: /contracts/SpeedBumpPriceGate.sol    #5
```

```
58                return gate.lastPrice - decay;
```

https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/SpeedBumpPriceGate.sol#L58

## [13] Remove unused local variable

```
File: /contracts/FixedPricePassThruGate.sol   #1

53                (bool sent, bytes memory data) = gate.beneficia
```

https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/FixedPricePassThruGate.sol#L53

```
File: /contracts/SpeedBumpPriceGate.sol   #2

79                (bool sent, bytes memory data) = gate.beneficia
```

https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/SpeedBumpPriceGate.sol#L79

## [14] Using `bool` s for storage incurs overhead

```
    // Booleans are more expensive than uint256 or any type that
    // word because each write operation emits an extra SLOAD to
    // slot's contents, replace the bits taken up by the boolear
    // back. This is the compiler's defense against contract upg
    // pointer aliasing, and it cannot be disabled.
```

https://github.com/OpenZeppelin/openzeppelin-contracts/blob/58f635312aa21f947cae5f8578638a85aa2519f5/contracts/security

/ReentrancyGuard.sol#L23-L27 Use `uint256(1)` and `uint256(2)` for true/false

```
File: /contracts/MerkleDropFactory.sol    #1

29        mapping (address => mapping (uint => bool)) public witk
```

https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/MerkleDropFactory.sol#L29

```
File: /contracts/MerkleResistor.sol    #2

50        mapping (address => mapping (uint => bool)) public init
```

https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/MerkleResistor.sol#L50

```
File: /contracts/MerkleVesting.sol    #3

38        mapping (address => mapping (uint => bool)) public init
```

https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/MerkleVesting.sol#L38

```
File: /contracts/VoterID.sol    #4

21        mapping (address => mapping (address => bool)) public c
```

https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/VoterID.sol#L21

## [15] `public` library function should be made `private` / `internal`

Changing from `public` will remove the compiler-introduced checks for `msg.value` and decrease the contract's method ID table size

```
File: contracts/MerkleLib.sol   #1

36      function parentHash(bytes32 a, bytes32 b) public pure re
```

https://github.com/code-423n4/2022-05-factorydao/blob/db415804c06143d8af6880bc4cda7222e5463c0e/contracts/MerkleLib.sol#L36

## [16] Move `if-else` to inside function call to save deployment gas

```
File: contracts/MerkleLib.sol   #1

37          if (a < b) {
38              return keccak256(abi.encode(a, b));
39          } else {
40              return keccak256(abi.encode(b, a));
41          }
```

change to `return keccak256(a < b ? abi.encode(a, b) : abi.encode(b, a)`
https://github.com/code-423n4/2022-05-factorydao/blob/db415804c06143d8af6880bc4cda7222e5463c0e/contracts/MerkleLib.sol#L37-L41

## [17] Use a more recent version of solidity

Use a solidity version of at least 0.8.10 to have external calls skip contract existence checks if the external call has a return value

```
File: /contracts/FixedPricePassThruGate.sol   #1
```

```
3   pragma solidity 0.8.9;
```

[https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/FixedPricePassThruGate.sol#L3](https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/FixedPricePassThruGate.sol#L3)

```
File: /contracts/interfaces/IVoterID.sol   #2

3   pragma solidity 0.8.9;
```

[https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/interfaces/IVoterID.sol#L3](https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/interfaces/IVoterID.sol#L3)

```
File: /contracts/MerkleDropFactory.sol   #3

3   pragma solidity 0.8.9;
```

[https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/MerkleDropFactory.sol#L3](https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/MerkleDropFactory.sol#L3)

```
File: /contracts/MerkleEligibility.sol   #4

3   pragma solidity 0.8.9;
```

[https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/MerkleEligibility.sol#L3](https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/MerkleEligibility.sol#L3)

```
File: /contracts/MerkleIdentity.sol   #5

3   pragma solidity 0.8.9;
```

https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/MerkleIdentity.sol#L3

```
File: /contracts/MerkleLib.sol    #6

3   pragma solidity 0.8.9;
```

https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/MerkleLib.sol#L3

```
File: /contracts/MerkleResistor.sol    #7

3   pragma solidity 0.8.9;
```

https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/MerkleResistor.sol#L3

```
File: /contracts/MerkleVesting.sol    #8

3   pragma solidity 0.8.9;
```

https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/MerkleVesting.sol#L3

```
File: /contracts/PermissionlessBasicPoolFactory.sol    #9

3   pragma solidity 0.8.9;
```

https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/PermissionlessBasicPoolFactory.sol#L3

```
File: /contracts/SpeedBumpPriceGate.sol   #10

3   pragma solidity 0.8.9;
```

[https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/SpeedBumpPriceGate.sol#L3](https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/SpeedBumpPriceGate.sol#L3)

```
File: /contracts/VoterID.sol   #11

3   pragma solidity 0.8.9;
```

[https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/VoterID.sol#L3](https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/VoterID.sol#L3)

```
File: /interfaces/IVoterID.sol   #12

3   pragma solidity 0.8.9;
```

[https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/interfaces/IVoterID.sol#L3](https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/interfaces/IVoterID.sol#L3)

## [18] It costs more gas to initialize variables to zero than to let the default of zero be applied

```
File: /contracts/MerkleLib.sol   #1

22          for (uint i = 0; i < proof.length; i += 1) {
```

[https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/MerkleLib.sol#L22](https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/MerkleLib.sol#L22)

```
File: /contracts/MerkleResistor.sol   #2

176           uint currentWithdrawal = 0;
```

[https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/MerkleResistor.sol#L176](https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/MerkleResistor.sol#L176)

```
File: /contracts/MerkleVesting.sol   #3

150           uint currentWithdrawal = 0;
```

[https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/MerkleVesting.sol#L150](https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/MerkleVesting.sol#L150)

```
File: /contracts/PermissionlessBasicPoolFactory.sol   #4

115           for (uint i = 0; i < rewardTokenAddresses.length;
```

[https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/PermissionlessBasicPoolFactory.sol#L115](https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/PermissionlessBasicPoolFactory.sol#L115)

```
File: /contracts/PermissionlessBasicPoolFactory.sol   #5

141           for (uint i = 0; i < pool.rewardFunding.length; i+
```

[https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/PermissionlessBasicPoolFactory.sol#L141](https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/PermissionlessBasicPoolFactory.sol#L141)

```
File: /contracts/PermissionlessBasicPoolFactory.sol   #6
```

```
168             for (uint i = 0; i < pool.rewardsWeiPerSecondPerT
```

https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/PermissionlessBasicPoolFactory.sol#L168

```
File: /contracts/PermissionlessBasicPoolFactory.sol   #7

224             for (uint i = 0; i < rewards.length; i++) {
```

https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/PermissionlessBasicPoolFactory.sol#L224

```
File: /contracts/PermissionlessBasicPoolFactory.sol   #8

249             for (uint i = 0; i < pool.rewardTokens.length; i+
```

https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/PermissionlessBasicPoolFactory.sol#L249

```
File: /contracts/PermissionlessBasicPoolFactory.sol   #9

266             for (uint i = 0; i < pool.rewardTokens.length; i+
```

https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/PermissionlessBasicPoolFactory.sol#L266

## [19] `++i` costs less gas than `++i`, especially when it's used in `for`-loops (`--i`/`i--` too)

Saves 6 gas *PER LOOP*

```
File: /contracts/PermissionlessBasicPoolFactory.sol    #1

115            for (uint i = 0; i < rewardTokenAddresses.length;
```

[https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/PermissionlessBasicPoolFactory.sol#L115](https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/PermissionlessBasicPoolFactory.sol#L115)

```
File: /contracts/PermissionlessBasicPoolFactory.sol    #2

141            for (uint i = 0; i < pool.rewardFunding.length; i+
```

[https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/PermissionlessBasicPoolFactory.sol#L141](https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/PermissionlessBasicPoolFactory.sol#L141)

```
File: /contracts/PermissionlessBasicPoolFactory.sol    #3

168            for (uint i = 0; i < pool.rewardsWeiPerSecondPerTo
```

[https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/PermissionlessBasicPoolFactory.sol#L168](https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/PermissionlessBasicPoolFactory.sol#L168)

```
File: /contracts/PermissionlessBasicPoolFactory.sol    #4

224            for (uint i = 0; i < rewards.length; i++) {
```

[https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/PermissionlessBasicPoolFactory.sol#L224](https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/PermissionlessBasicPoolFactory.sol#L224)

```
File: /contracts/PermissionlessBasicPoolFactory.sol    #5
```

```
249            for (uint i = 0; i < pool.rewardTokens.length; i+
```

[https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/PermissionlessBasicPoolFactory.sol#L249](https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/PermissionlessBasicPoolFactory.sol#L249)

```
File: /contracts/PermissionlessBasicPoolFactory.sol   #6

266            for (uint i = 0; i < pool.rewardTokens.length; i+
```

[https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/PermissionlessBasicPoolFactory.sol#L266](https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/PermissionlessBasicPoolFactory.sol#L266)

```
File: contracts/MerkleLib.sol   #7

22           for (uint i = 0; i < proof.length; i += 1) {
```

[https://github.com/code-423n4/2022-05-factorydao/blob/db415804c06143d8af6880bc4cda7222e5463c0e/contracts/MerkleLib.sol#L22](https://github.com/code-423n4/2022-05-factorydao/blob/db415804c06143d8af6880bc4cda7222e5463c0e/contracts/MerkleLib.sol#L22)

## [20] Using `private` rather than `public` for constants, saves gas

If needed, the value can be read from the verified contract source code. Savings are due to the compiler not having to create non-payable getter functions for deployment calldata, and not adding another entry to the method ID table

```
File: /contracts/MerkleResistor.sol   #1

59       uint constant public PRECISION = 1000000;
```

https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/Mer kleResistor.sol#L59

## [21] Duplicated `require()` / `revert()` checks should be refactored to a modifier or function

Saves deployment costs

```
File: /contracts/PermissionlessBasicPoolFactory.sol   #1

182           require(pool.id == poolId, 'Uninitialized pool');
```

https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/PermissionlessBasicPoolFactory.sol#L182

```
File: /contracts/PermissionlessBasicPoolFactory.sol   #2

234           require(success, 'Token transfer failed');
```

https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/PermissionlessBasicPoolFactory.sol#L234

## [22] Stack variable used as a cheaper cache for a state variable is only used once

If the variable is only accessed once, it’s cheaper to use the state variable directly that one time

```
File: /contracts/VoterID.sol   #1

152           address oldOwner = _owner_;
```

https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/VoterID.sol#L152

## [23] `require()` or `revert()` statements that check input arguments should be at the top of the function

Checks that involve constants should come before checks that involve state variables

```
File: /contracts/PermissionlessBasicPoolFactory.sol   #1

316           require(newTaxPerCapita < 1000, 'Tax too high');
```

https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/PermissionlessBasicPoolFactory.sol#L316

```
File: contracts/PermissionlessBasicPoolFactory.sol   #2

112          require(rewardsWeiPerSecondPerToken.length == rewar
```

https://github.com/code-423n4/2022-05-factorydao/blob/db415804c06143d8af6880bc4cda7222e5463c0e/contracts/PermissionlessBasicPoolFactory.sol#L112

```
File: contracts/PermissionlessBasicPoolFactory.sol   #3

159          require(pool.id == poolId, 'Uninitialized pool');
```

https://github.com/code-423n4/2022-05-factorydao/blob/db415804c06143d8af6880bc4cda7222e5463c0e/contracts/PermissionlessBasicPoolFactory.sol#L159

## [24] Use custom errors rather than `revert()` / `require()` strings to save deployment gas

Custom errors are available from solidity version 0.8.4. The instances below match or exceed that version

See original submission for details.

## [25] `public` functions not called by the contract should be declared `external` instead

Contracts are allowed to override their parents' functions and change the visibility from `external` to `public` and can save gas by doing so.

```
File: /contracts/MerkleDropFactory.sol   #1

49       function addMerkleTree(bytes32 newRoot, bytes32 ipfsHas
```

https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/MerkleDropFactory.sol#L49

```
File: /contracts/MerkleDropFactory.sol   #2

88       function withdraw(uint treeIndex, address destination,
```

https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/MerkleDropFactory.sol#L88

```
File: /contracts/MerkleIdentity.sol   #3

140       function getPrice(uint merkleIndex) public view returr
```

https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/MerkleIdentity.sol#L140

```
File: /contracts/MerkleIdentity.sol   #4

152       function isEligible(uint merkleIndex, address recipier
```

https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/MerkleIdentity.sol#L152

```
File: /contracts/MerkleLib.sol   #5

17       function verifyProof(bytes32 root, bytes32 leaf, bytes3
```

https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/MerkleLib.sol#L17

```
File: /contracts/MerkleResistor.sol   #6

80       function addMerkleTree(bytes32 newRoot, bytes32 ipfsHas
```

https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/MerkleResistor.sol#L80

```
File: /contracts/MerkleVesting.sol   #7

62       function addMerkleRoot(bytes32 newRoot, bytes32 ipfsHas
```

https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/MerkleVesting.sol#L62

```
File: /contracts/VoterID.sol   #8

270       function isApprovedForAll(address _address, address op
```

https://github.com/code-423n4/2022-05-factorydao/blob/e22a562c01c533b8765229387894cc0cb9bed116/contracts/VoterID.sol#L270

illuzen (FactoryDAO) commented:

> All duplicates except
> 2: valid, I guess, but negligible benefit
> 14: valid, but negligible
> 15: valid
> 20: valid

Justin Goro (judge) commented:

> Issue 21 is QA and is an opinion rather than a clear improvement. The dev may have had reasons for going the route they did.

> Note to wardens: for deployment gas improvements, unless the improvements are significant and impact final contract size significantly, these optimizations are the least important since they have no impact on the end user of the contract.

Justin Goro (judge) commented:

> There were 25 improvements reported. An impact score has been assigned to each. 0 indicates either that the impact is negligible, the improvement is invalid or the item is not actually a gas optimization. 1 indicates a valid but low or negligible improvement and 2 indicates a significant improvement worth serious consideration.

| | Title | Impact |
|---|---|---|
| 1 | `for` -loops should be broken out of earlier | 2 |
| 2 | Multiple `address` mappings can be combined into a single `mapping` of an `address` to a `struct` , where appropriate | 2 |
| 3 | State variables only set in the constructor should be declared `immutable` | 1 |
| 4 | Using `calldata` instead of `memory` for read-only arguments in `external` functions saves gas | 1 |
| 5 | State variables should be cached in stack variables rather than re-reading them from storage | 1 |
| 6 | `<x> += <y>` costs more gas than `<x> = <x> + <y>` for state variables | 1 |
| 7 | `internal` functions only called once can be inlined to save gas | 1 |

|  | Title | Impact |
|---|---|---|
| 8 | Add `unchecked {}` for subtractions where the operands cannot underflow because of a previous `require()` | 1 |
| 9 | `<array>.length` should not be looked up in every loop of a `for` -loop | 2 |
| 10 | `++i` / `i++` should be `unchecked{++i}` / `unchecked{++i}` when it is not possible for them to overflow, as is the case when used in `for` - and `while` -loops | 1 |
| 11 | `require()` / `revert()` strings longer than 32 bytes cost extra gas | 2 |
| 12 | Not using the named return variables when a function returns, wastes deployment gas | 0 |
| 13 | Remove unused local variable | 1 |
| 14 | Using `bool` s for storage incurs overhead | 2 |
| 15 | `public` library function should be made `private` / `internal` | 1 |
| 16 | Move `if-else` to inside function call to save deployment gas | 0 |
| 17 | Use a more recent version of solidity | 2 |
| 18 | It costs more gas to initialize variables to zero than to let the default of zero be applied | 1 |
| 19 | `++i` costs less gas than `++i` , especially when it's used in `for` -loops ( `--i` / `i--` too) | 2 |
| 20 | Using `private` rather than `public` for constants, saves gas | 1 |
| 21 | Duplicated `require()` / `revert()` checks should be refactored to a modifier or function | 0 |
| 22 | Stack variable used as a cheaper cache for a state variable is only used once | 0 |
| 23 | `require()` or `revert()` statements that check input arguments should be at the top of the function | 1 |
| 24 | Use custom errors rather than `revert()` / `require()` strings to save deployment gas | 2 |
| 25 | `public` functions not called by the contract should be declared `external` instead | 1 |

## Disclosures

C4 is an open organization governed by participants in the community.

C4 Contests incentivize the discovery of exploits, vulnerabilities, and bugs in smart contracts. Security researchers are rewarded at an increasing rate for finding higher-risk issues. Contest submissions are judged by a knowledgeable security researcher and solidity developer and disclosed to sponsoring developers. C4 does not conduct formal verification regarding the provided code but instead provides final verification.

C4 does not provide any guarantee or warranty regarding the security of this project. All smart contract software should be used at the sole risk and responsibility of users.

Top