·Decurity·

Smart Contract Security Audit Report

1inch Farming

Decurity, 2022

Contents

1. General Information

This report contains information about the results of the security audit of the 1inch (hereafter referred to as “Customer”) smart contracts, conducted by Decurity in the period from 11/22/2022 to 12/02/2022.

1.1. 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.

1.2. Scope of Work

The audit scope included the contracts in the following repository: https://github.com/1inch/farming. Initial review was done for the commit 8403ce63df903db66c5c8fd7958470c6827df7e6 and the re-testing was done for the commit 69595bbe9c0feedfb9017dc3f167dd9007ff9e88.

1.3. Threat Model

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

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

1.4. 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. Summary

As a result of this work, we have discovered a single critical exploitable security issue which has been fixed and re-tested in the course of the work.

The other suggestions included fixing the low-risk issues and some best practices (see 3.1).

The 1inch team has given the feedback for the suggested changes and explanation for the underlying code.

2.1. Suggestions

The table below contains the discovered issues, their risk level, and their status as of Dec 2, 2022 .

Table. Discovered weaknesses

Issue	Contract	Risk Level	Status
Variable truncation	accounting/FarmAccounting.sol	Medium	Fixed
Fees are not checked for transfers	FarmingPool.sol FarmingPod.sol MultiFarmingPod.sol	Medium	Acknowledged
User can take borrow stakingToken and increase total supply of FarmingPool without collateral in case of using ERC-777	FarmingPool.sol	Medium	Acknowledged
The logic of reward payout may be broken by rebasing tokens	accounting/FarmingAccount.sol	Medium	Acknowledged

The distributor can borrow stakingToken in the rescueFunds in case of using ERC-777	FarmingPool.sol	**Low**	Acknowledged
Cache array length outside the loop	MultiFarmingPod.sol	**Info**	Fixed
Non-optimal addition/subtraction	FarmingPod.sol MultiFarmingPod.sol /accounting/UserAccounting.sol	**Info**	Acknowledged
Split if statement that using &&	FarmingPool.sol /accounting/UserAccounting.sol	**Info**	Acknowledged
Missing 0x0 check on input addresses	FarmingPod.sol MultiFarmingPod.sol FarmingPool.sol	**Info**	Fixed
Variable shadowing	FarmingPod.sol MultiFarmingPod.sol	**Info**	Fixed
Literal value is used instead of constant	/accounting/FarmAccounting.sol	**Info**	Acknowledged

3. General Recommendations

This section contains general recommendations on how to fix discovered weaknesses and vulnerabilities and how to improve overall security level.

Section 3.1 contains a list of general mitigations against the discovered weaknesses, technical recommendations for each finding can be found in section 4.

Section 3.2 describes a brief long-term action plan to mitigate further weaknesses and bring the product security to a higher level.

3.1. Current Findings Remediation

Follow the recommendations in section 4.

3.2. Security Process Improvement

- 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.

4. Findings

4.1. Variable truncation

Risk Level: Medium

Contracts:

- accounting/FarmAccounting.sol

Status: Fixed in the commit https://github.com/1inch/farming/commit/53a0bc07587892a17eaf934c02a2bd805f1690a1.

Description:

Commit bda25f changed the position of the validation code block in FarmAccounting.sol:

```
    function startFarming(Info storage info, uint256 amount, uint256
period) internal returns(uint256) {
+       if (period == 0) revert ZeroDuration();
+       if (period > type(uint32).max) revert DurationTooLarge();
+       if (amount > _MAX_REWARD_AMOUNT) revert AmountTooLarge();

        // If something left from prev farming add it to the new farming
        Info memory prev = info;
        if (block.timestamp < prev.finished) {
            amount += prev.reward - farmedSinceCheckpointScaled(prev,
prev.finished - prev.duration) / _SCALE;
        }

-       if (period == 0) revert ZeroDuration();
-       if (period > type(uint32).max) revert DurationTooLarge();
-       if (amount > _MAX_REWARD_AMOUNT) revert AmountTooLarge();
```

```
      (info.finished, info.duration, info.reward) = (uint40(block.timestamp
+ period), uint32(period), uint184(amount));
       return amount;
```

In the previous implementation the validation of `amount` took into account the condition when there are some tokens left from the previous farming period so that total `amount` does not exceed `_MAX_REWARD_AMOUNT`. However, the new commit introduced a bug that may allow the `amount` to exceed `_MAX_REWARD_AMOUNT` since the check is done before `amount += prev.reward`. As a result the total `amount` might be truncated after `uint184(amount)` and an incorrect reward amount will be stored in the `Info` struct.

Remediation:

Check that `amount > _MAX_REWARD_AMOUNT` after the `amount` is summed with previous rewards.

4.2. Fees are not checked for transfers

Risk Level: Medium

Contracts:

- FarmingPool.sol
- FarmingPod.sol
- MultiFarmingPod.sol

References:

- https://github.com/compound-finance/compound-protocol/blob/6548ec3e8c733d18cd4bdb2c21197f666f828f35/contracts/CErc20.sol#L156

Status: Won't fix. Tokens with fees are not supported.

Description:

The argument `amount` should not be used as a trusted value in functions. This argument may not reflect the actual value being transferred (e.g. when there is a fee on transfer).

There are 4 instances of this issue:

```
     contracts/FarmingPod.sol:
       59:         rewardsToken.safeTransferFrom(msg.sender, address(this),
amount);

     contracts/FarmingPool.sol:
       62:         rewardsToken.safeTransferFrom(msg.sender, address(this),
amount);
       83:         stakingToken.safeTransferFrom(msg.sender, address(this),
amount);

     contracts/MultiFarmingPod.sol:
       73:         rewardsToken.safeTransferFrom(msg.sender, address(this),
amount);
```

Remediation:

Consider checking the actual amount received by the contract after the transfer function has been executed.

4.3. User can borrow stakingToken and increase total supply of FarmingPool without collateral

Risk Level: Medium

Contracts:

- FarmingPool.sol

References:

- https://swcregistry.io/docs/SWC-107
- https://dasp.co/#item-1

Status: Noted. Risk accepted.

Description:

When the stakingToken is ERC-777, anyone can use the deposit() function to borrow a free flash loan of the stakingToken from FarmingPool.

There will be the following sequence:

```
→ user calling deposit() of the FarmingPool
    →stakingToken.safeTransferFrom(user, farm, amount)
        → user.tokensToSend() // ERC777 pre-transfer hook
            → FarmingPool.withdraw()
            → stakingToken.safeTransfer() // repay flash loan
```

In case when a user calls a deposit with an amount equal to N after withdrawing their balance will be doubled (2*N).

Remediation:

Follow the checks-effects-interactions pattern to prevent exploitation of the re-entrancy attack.

```
      function deposit(uint256 amount) external override {
              stakingToken.safeTransferFrom(msg.sender, address(this),
amount);
              _mint(msg.sender, amount);
      }
```

4.4. The logic of reward payout may be broken by rebasing tokens

Risk Level: Medium

Contracts:

- FarmAccounting.sol

Status: Won't fix. Rebasing tokens are not supported.

Description:

Due to a lack of a check that rewards token balances can change, rebasable tokens can cause the calculation of farmed rewards in "farmedSinceCheckpointScaled()" to mess up.

```
      contracts/accounting/FarmAccounting.sol:
         27:                        return  elapsed  *  info.reward  *  _SCALE  /
info.duration;
```

Remediation:

Make sure that the property is documented if they are unsupported.

4.5. The distributor can borrow stakingToken in the rescueFunds in case of using ERC-777

Risk Level: Low

Contracts:

- FarmingPool.sol

References:

- https://swcregistry.io/docs/SWC-107
- https://dasp.co/#item-1

- https://consensys.net/diligence/audits/2020/12/1inch-liquidity-protocol/#the-owner-can-borrow-token0token1-in-the-rescuefunds

Status: Won't fix.

Description:

When the stakingToken is ERC-777, the distributor can use the `rescueFunds()` function to borrow a free flash loan of the stakingToken from FarmingPool.

Remediation:

Consider checking the amount of rescuing tokens before making a transfer:

```
      function    rescueFunds(IERC20    token,    uint256    amount)    external
onlyDistributor {
          if (token == IERC20(address(0))) {
              payable(distributor).sendValue(amount);
          } else {
              if (token == stakingToken) {
                       if  (stakingToken.balanceOf(address(this))  -  amount  <
totalSupply()) revert NotEnoughBalance();
              }
              token.safeTransfer(distributor, amount);
          }
      }
```

4.6. Cache array length outside the loop

Risk Level: Info

Contracts:

- MultiFarmingPod.sol

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

Status: Fixed in the commit https://github.com/1inch/farming/commit/ee7a4bd7d2655800a22eb450fd12125bd47b0198.

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:

```
    MultiFarmingPod.sol::103   =>   for   (uint256   i   =   0;   i   <
tokens.length; i++) {
    MultiFarmingPod.sol::112   =>   for   (uint256   i   =   0;   i   <
tokens.length; i++) {
```

Remediation:

Example of a not optimized code:

```
for (uint256 i = 0; i < array.length; i++) {
    // invariant: array's length is not changed
}
```

Consider saving array length before the loop:

```
uint256 len = array.length;
for (uint256 i = 0; i < len; i++) {
    // invariant: array's length is not changed
}
```

4.7. Non-optimal addition/subtraction

Risk Level: Info

Contracts:

- FarmingPod.sol
- MultiFarmingPod.sol
- /accounting/UserAccounting.sol

References:

- https://gist.github.com/IllIllI000/cbbfb267425b898e5be734d4008d4fe8

Status: Won't fix for readability.

Description:

Using the addition/submission operator instead of plus-equals/minus-equals saves gas.

There are the following cases:

```
contracts/FarmingPod.sol:
  80:             _totalSupply += amount;
  83:             _totalSupply += amount;

contracts/MultiFarmingPod.sol:
  117:             _totalSupply += amount;
  120:             _totalSupply += amount;

contracts/accounting/UserAccounting.sol:
  54:              info.corrections[from] -= diff;
  57:              info.corrections[to] += diff;
```

Remediation:

Example of a not optimized code:

```
    issuedSupply += amount;
```

Consider using addition/submission operators:

```
    issuedSupply = issuedSupply + amount;
```

4.8. Split if statement that is using && logical operator

Risk Level: Info

Contracts:

- FarmingPool.sol
- /accounting/UserAccounting.sol

References:

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

Status: Won't fix for readability.

Description:

Gas costs are higher when using && logical operator.

Remediation:

Example of a not optimized code:

```
    if (amount > 0 && from != to) {
                userInfo.updateBalances(from,        to,        amount,
farmedPerToken());
    }
```

Consider separate check:

```
    if (amount > 0) {
         if(from != to) {
                userInfo.updateBalances(from,        to,        amount,
farmedPerToken());
```

```
                    }
    }
```

4.9. Missing 0x0 check on input addresses

Risk Level: Info

Contracts:

- FarmingPod.sol
- MultiFarmingPod.sol
- FarmingPool.sol

Status: Fixed in the commit https://github.com/1inch/farming/commit/69595bbe9c0feedfb9017dc3f167dd9007ff9e88.

Description:

There are multiple occurrences of dangerous assignment of the input address to a storage variable without proper sanitization, namely not checking that the address is zero.

There are the following occurrences:

- Variable distributor_ in setDistributor in FarmingPod.sol (line 51)
- Variable distributor_ in setDistributor in MultiFarmingPod.sol (line 58)
- Variable rewardsToken in addRewardsToken in MultiFarmingPod.sol (line 65)
- Variable distributor_ in setDistributor in FarmingPool.sol (line 55)

Remediation:

Always check that an input address is not zero before assigning a storage variable.

4.10. Variable shadowing

Risk Level: Info

Contracts:

- FarmingPod.sol
- MultiFarmingPod.sol

Status: Fixed in the commit https://github.com/1inch/farming/commit/f42ffd23c5d7b850676422da0fddb366e2f52608.

Description:

The variable `token` in the `rescueFunds()` function shadows the state variable `token` inherited from the Pod contract.

There are the following occurrences:

- FarmingPod.sol#L87
- MultiFarmingPod.sol#L124

Remediation:

To prevent shadowing, consider renaming the `token` variable in the `rescueFunds()` function.

4.11. Literal value is used instead of constant

Risk Level: Info

Contracts:

- /accounting/FarmAccounting.sol

Status: Won't fix.

Description:

There is a period check:

```
contracts/accounting/FarmAccounting.sol:
39: if (period > type(uint32).max) revert DurationTooLarge();
```

It is a good practice to store such values in constants, making them declarative and reusable.

Remediation:

Consider storing `type(uint32).max` in a constant, like it was done with `_MAX_REWARD_AMOUNT`:

```
    uint256 constant internal _MAX_REWARD_AMOUNT = 1e42;
```

5. Appendix

5.1. About us

The Decurity (former DeFiSecurity.io) 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.